June 2014

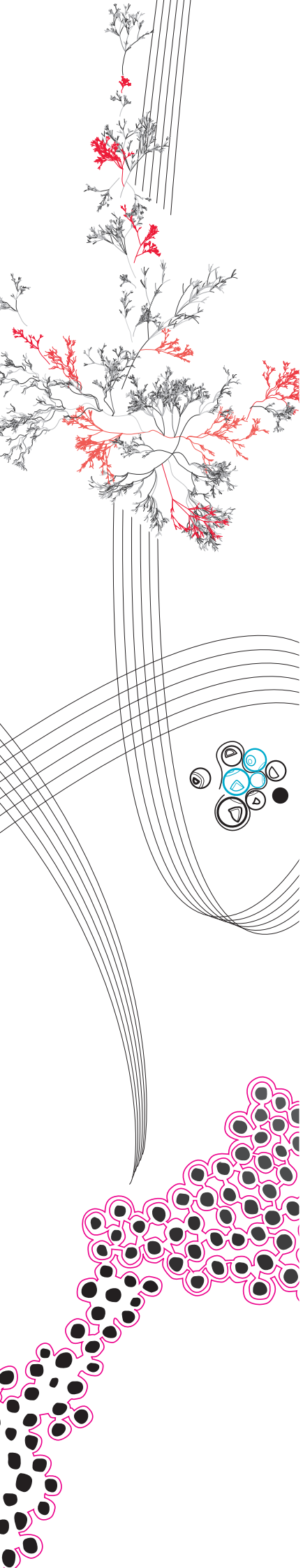
MASTER THESIS

ADDING QUALITY OF SERVICE TO DISRUPTION TOLERANT NETWORKING USING THE ORWAR ROUTING PROTOCOL FOR USE IN MILITARY MANETS

Gertjan Kamphuis

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
Design and Analysis of Communication Systems (DACS)
In cooperation with Thales Nederland

Exam committee:
dr. ir. G.J. Heijenk
prof. dr. J.L. van den Berg
dr. ir. J. Goseling
dr. ir. M. de Graaf (Thales Nederland)
dr. ir. G.J. Hoekstra (Thales Nederland)



UNIVERSITY OF TWENTE.

Abstract

As armed forces become more technologically advanced, so does the need to effectively use this technology increase. One of the developments within the military for the past years is its increasing networking capabilities. Mobile ad-hoc networks are used in the military environment to increase cohesiveness between units and improve their efficiency. These networks however have to operate in hostile environments where there might not always be a (viable) direct connection between endpoints. Disruption Tolerant Networking (DTN) can help alleviate this issue by offering means to transfer data between endpoints using Store-Carry-Forward message switching, eliminating the need for end-to-end connectivity. To facilitate routing on these networks, several protocols have been developed.

One of these protocols is the Opportunistic Routing with Window-Aware Replication (ORWAR) protocol. This protocol aims to optimize power and bandwidth use by dynamically reordering its transmission buffer and limiting incomplete transmissions. It also uses a priority scheduling system in an attempt to provide limited Quality of Service (QoS) to the network. In this thesis we further analyse this protocol and aim to improve upon it to better provide QoS provisioning for military networks.

To this purpose, we determine the requirements traffic on a military network imposes on the network. Using these requirements, we assign all traffic a transmission priority and retention priority class and adapt the ORWAR protocol to provide support for these classes. We then analyse whether this division into two traffic classes has a positive influence on network performance, using a mobility model which accurately reflects the military environment.

Acknowledgements

Before you lies the thesis which marks the end of my master's programme Telematics at the University of Twente. A place where I have significantly developed myself as a person and it is with a heavy heart I will say goodbye to it. However, first, before proceeding any further I would like to thank a few people and instances which made all this work possible

First of all, I would like to thank Thales Nederland for providing me with the opportunity to do my research at their offices in Huizen. It has been a wonderful experience to work there, with a very warm and welcoming atmosphere.

I would also like to express my gratitude to dr. ir. M. de Graaf and dr. ir. G.J. Hoekstra for their excellent support during my time at Thales. Their views, comments and support have been a tremendous help in producing this work. Of course I would also like to thank the other members of my committee: dr. ir. G.J. Heijenk, dr. ir. J. Goseling and prof. dr. J.L. van den Berg. Without their feedback on my research, this thesis would not have been possible.

Furthermore, I would like to thank my lab partners at Thales, Javier Cuadros Linde and Shyam Balasubramanian. Our discussions on the inner workings of DTN and ORWAR along with a fresh view on my problems have helped me greatly in producing these results. In addition, the lab was never a dull moment with you two.

Special thanks also go out to my girlfriend, who has supported me all this time and motivated me when the going got tough. Without you, producing these results would have been a lot harder.

Lastly, I of course would like to thank all my family and friends which have supported me these past years. You are the ones that made this journey all worthwhile!

Contents

List of Acronyms

BC	Buffer Contents
BFT	Blue Force Tracking
BMS	Battlefield Management System
C2	Command & Control
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
CTS	Clear To Send
DNS	Domain Name System
DTN	Disruption/Delay Tolerant Networking
EID	Endpoint ID
GPS	Global Positioning System
KDM	Known Delivered Messages
MANET	Mobile Ad-hoc Network
ORWAR	Opportunistic Routing with Window Aware Replication
PBB	Primary Bundle Block
PDU	Protocol Data Unit
PRoPHET	Probabilistic Routing Protocol using History of Encounters and Transitivity
QoS	Quality of Service
RAPID	Resource Allocation Protocol for Intentional DTN routing
RID	Region ID
RTS	Request To Send
SCF	Store Carry Forward
SSA	Shared Situational Awareness
TDMA	Time Division Multiple Access
TTL	Time To Live
WAN	Wide Area Network

Chapter 1

Introduction

1.1 Background

The time where armies fought each other on isolated battlefields, led by their local commanders with little communication between units has long passed. Over the years, as technology developed, so did communication become more and more important. This development has evolved to a point where battles can be won or lost through the communication capabilities of armed forces. This concept is called *information superiority*, defined by[1]:

> The ability to collect, process, and disseminate an uninterrupted flow of information while exploiting and/or denying an adversary's ability to do the same.

The concept of information superiority has led to the development of *Network Centric Warfare*(NCW). A type of combat characterized by the need to obtain information superiority over the opponent, NCW describes ways a fully or partially networked force can gain a decisive advantage over its opponent.

In the light of NCW, a lot of research has been performed in improving connectivity in military environments. Nowadays almost every unit is connected with other units one way or the other, using a wide range of communication technologies; these include radio, wifi and satellite communications. These connections result in a wide-scale network which has both components with and without infrastructure, e.g. networks on a base vs. wireless connections between frontline troops.

On the battlefield, vehicles and troops are sparsely connected over large areas and in general have limited resources available to them. In addition, there generally is no fixed infrastructure available to these mobile nodes. Because these nodes are constantly on the move, with no fixed infrastructure, wireless transmissions in a Mobile Ad-hoc Networking (MANET) configuration are used. This lack of infrastructure results in the need for special routing and discovery algorithms to provide a connection between two nodes.

In addition to these extra algorithms, the mobility of the nodes results in the network being very dynamic, with some connections only being intermittently available. On the battlefield these problems are amplified as enemy forces will actively try to disrupt communications in any way possible or nodes will be unreachable due to the need of radio silence. An example of all the problems summarized here can be seen in Figure 1.1.

Disruption Tolerant Networking (DTN)[2][3] can help alleviate these issues. DTN networks are networks characterized by communications between two nodes possibly taking a longer period of time

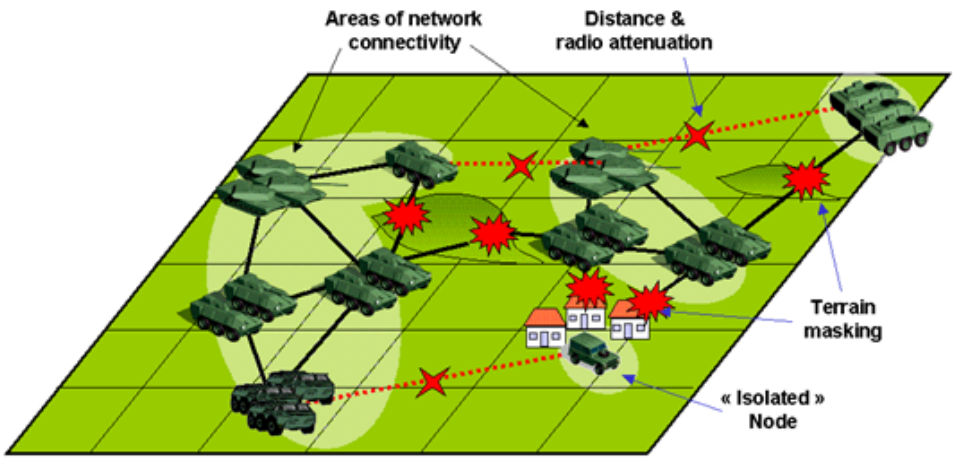


Figure 1.1: MANET issues in military environments

than expected in contemporary wireless networks. These networks are specialized in dealing with not always having end-to-end connectivity and/or long delays which will inevitably break down ordinary connection-oriented protocols such as TCP.

DTN employs *Store Carry Forward* (SCF) message switching to disseminate its messages across the network. This means copies of messages are transmitted to intermittent nodes on the network. These will then store the message copy, carry it with them while they move, and forward it when they meet another node. Furthermore, DTN is able to transmit messages regardless of lower-layer transmission methods. It is therefore possible for a message to be partially transmitted using TCP/IP connections and partially through proprietary lower-layer protocols.

1.2 Disruption Tolerant Networking

Originally developed as a means to provide inter-planetary networking (IPN)[4], DTN addresses some issues of modern mobile wireless networks which conflicts with some of the original assumptions of the internet. These issues include *intermittent connectivity*, *long or variable delays*, *asymmetric data rates* and *high error rates*. In intermittent connectivity, links between nodes appear and disappear, this can result in no end-to-end connectivity existing between two specific nodes, also called *network partitioning*. In addition to intermittent connectivity, long propagation delays between nodes and variable queuing delays at nodes in a multi-hop network can increase the end-to-end path delay significantly. This way, internet protocols relying on a quick replies from other nodes will not function anymore.

To support connectivity between nodes communicating using a network suffering these issues, DTN introduces the *Bundle Protocol* [5]. This protocol operates between the Application layer and underlying transmission layers, providing SCF functionality to a node. Individual application messages will get stored in one or more bundles, which in turn will be transmitted using underlying transmission protocols. It is also in this *bundle layer* that DTN-specific routing occurs.

In contrast to normal internet routing, messages, or fragments thereof, are stored in persistent storage at a node rather than a limited-size and -time buffer. Using the SCF principle explained earlier, even if no end-to-end connection exists due to e.g. network partitioning, messages are still able to be delivered to their destination.

Due to the fact that DTN is independent of underlying transmission protocols, DTN nodes need not see each other directly. Instead of only propagating messages to next-hop neighbours running DTN, messages can also be transmitted through other nodes not running DTN. For instance, a DTN node connected to the internet, can transmit DTN messages to another DTN node connected to the internet, even though there may be intermittent hops which cannot process DTN messages. This however will be dependent on which DTN routing protocol is used as we will explain in Chapter 2

In traditional MANET routing protocols like OLSR[6] and AODV[7] a packet traverses the network by being forwarded from one hop to the other. These protocols can be either proactive, reactive or a hybrid of the former two. A proactive protocol, such as OLSR, maintains a list of routes to possible destinations by occasionally distributing routing tables across the network. These protocols are generally slow to adapt to rapidly changing topologies. Reactive protocols, such as AODV, on the other hand only provide on-demand routing information using *Route REQuest* (RREQ) packets. While these protocols are better suited to rapidly changing topologies, the extra flooding of RREQ packets will waste bandwidth. In addition, these protocols may show high latency when attempting to discover a route to the destination.

These protocols do not always perform well in Disruption Tolerant Networks as they rely on end-to-end connectivity, which might not always be available. For this reason, a new kind of routing called *replication based routing* has been developed. Unlike traditional routing schemes where messages get forwarded to the next hop, in replication based routing a *copy* of the message will get forwarded to the next hop. This results in multiple copies of a message being available on the network, increasing the probability it will arrive at the intended destination. This routing is coupled with SCF to reduce the need for end-to-end connectivity.

One of the simplest forms of replication based routing is the flooding of message copies to every node within range. While this form of routing will almost guarantee the delivery of a message on the network, it is also very wasteful of resources, especially bandwidth. In Disruption Tolerant Networks, there is currently no single standardized routing protocol as different protocols aim to improve different metrics. Most Disruption Tolerant Networks serve a specific purpose or operate in a specific environment and therefore impose different requirements on the routing protocols. For instance, a network used for Inter-Planetary Internet may have different requirements than one used for military networks.

In addition to flooding, many different, more sophisticated, routing schemes have been researched to provide routing in Disruption Tolerant Networks, with some examples being [8][9][10][11][12]. In military MANETs, the main resource limitation is bandwidth. This is due to the enemy attempting to disrupt communications and imposed periods of Emissions Control (EMCON), also known as radio silence. These bandwidth limitations are in addition to the already limited bandwidth due to the sparsely connected nature of a military MANET.

Opportunistic Routing with Window-Aware Replication (ORWAR) [13] is one of these routing protocols, which aims to optimize bandwidth use and power control. This protocol tries to optimize bandwidth use and power control through a combination of limiting incomplete transmissions and priority scheduling of messages. For this purpose, it uses localization information of a network node, acquired through for instance GPS. This information contains a node's location, velocity and transmission range. Based on this information, it will calculate the time two nodes remain within transmission range of each other and schedule messages in such a way, no incomplete transfers occur due to nodes moving out of range. In [14], this protocol is deemed as the one most suited for a military MANET and the one we will try to improve upon in this research.

1.3 QoS in Disruption Tolerant Networks

In military networks, different traffic classes exist, imposing different requirements on the network. In contrast to commercial networks, stringent Quality of Service (QoS) requirements can be imposed on the different traffic types within a military network. Similarly, nodes on a military network can be forced to offer support for these QoS requirements. While QoS has been mentioned for DTN, there are no standards on how to facilitate QoS in these networks.

The only mention of QoS in DTN, as stated in [2], states that bundles can be assigned one of three priority classes: *bulk*, *normal* and *expedited*. These classes however only state how to differentiate bundles originating from the same source. This means that a high priority bundle from one source may not be delivered faster, or with some other superior QoS, than a medium priority bundle from a different source. In addition, depending on a particular DTN node's forwarding/scheduling policy, priority scheduling may or may not be enforced.

QoS in Disruption Tolerant Networks is also hard to guarantee due to the nature of these networks. Traffic is generally not interactive and is often one-way. There are generally no strong guarantees of timely delivery, though QoS may for instance be used to improve reliability and security. Different routing protocols attempt to offer some form of QoS due to their routing mechanics. The Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET) [9][15] for instance attempts to improve reliability by calculating the probability of a node meeting the destination. Further analysis of the different routing protocols and the way in which they attempt to improve QoS of network traffic is given in [14].

In this research, we focus on analysing what traffic classes military MANETs impose on DTN. Using this knowledge, we try to improve the ORWAR routing protocol by providing support for these QoS requirements. Finally, we analyse whether these QoS improvements actually improve performance.

1.4 Research Questions

This research focuses on improving the performance of DTN in a military environment using the ORWAR routing protocol. This should be achieved through implementing QoS support in this protocol. Our main research question therefore is:

Can we improve ORWAR to better support QoS provisioning for use in military MANETs?

To answer this question, the following sub-questions have been defined:

1. What traffic classes can be distinguished in military MANETs and what requirements do they impose?
2. Does the current design of ORWAR support a separation in transmission priority QoS and buffer management QoS?
3. How can the required QoS be implemented in ORWAR?
4. How does the implemented QoS perform in comparison to non-QoS ORWAR?

1.5 Thesis overview

The rest of this thesis is structured as follows: *Chapter 2* provides some background information on the military environment and the DTN protocols used. In specific, Section 2.1 provides an overview of the different traffic classes within a military network and their QoS requirements. These requirements are then translated into concrete QoS classes to be used for messages on a DTN network. Section 2.2 provides an overview of the bundle protocol, which has been designed to facilitate DTN communications on a network. Section 2.3 provides an overview of ORWAR and its workings.

Chapter 3 describes some of the limitations of the protocol which have to be addressed to be able to use it in a realistic military environment. Part of this research has been addressing these issues and offer solutions to enable the use of it for simulations. The limitations here could be split into general limitations of the protocol and arising issues occurring due to it not being designed to support communication between more than two nodes at a time. Improvements to enable communication between multiple nodes are introduced here as well.

Chapter 4 states which adaptations are needed to both the bundle protocol and ORWAR to provide for the QoS determined in Section 2.1.

Chapter 5 provides an overview of our experimental setup. A description of the different mobility models will be given, as well as network characteristics and our implemented adaptations of ORWAR. Furthermore, we describe our approach of how to compare the performance of normal ORWAR to our QoS-enabled ORWAR.

Chapters 6 and 7 detail the network performance measurement results using different mobility models. Section 6 mainly focuses on analysing the behaviour of ORWAR as we add more parameters using simplified mobility models. Section 7 uses a realistic military mobility model to compare the performance of normal ORWAR to the performance of QoS-enabled ORWAR.

Finally, *Chapter 8* will provide a summary of this work and the reached conclusions. In this section, the research questions will be answered and future work will be discussed.

Chapter 2

Background and Related work

2.1 QoS in military networks

2.1.1 Traffic types

Military MANETs are generally used to improve *Shared Situational Awareness* (SSA) and to communicate *Clear and Consistent Command Intent*. These are defined in [16, p.137] as

> The capability to extract meaningful activities and patterns from the battlespace picture and to share this awareness across the network with appropriate participants.

and

> The capability to articulate decisions in terms of desired goals/effects, constraints, and priorities that are functionally aligned across the network and with other participating organizations.

respectively.

For this purpose, nodes transmit *Command and Control*(C2) data to each other. Some examples of this data are:

- Voice communications
- Video streams, such as camera feeds from vehicles and personnel
- Targeting data
- *Blue Force Tracking* (BFT): location updates of allied forces

These datatypes bring their own requirements in terms of delay, reliability and bandwidth. Live camera feeds for instance need low delay and consume large amounts of bandwidth. A military MANET should be able to cope with these requirements and allocate resources correspondingly.

All the different C2 traffic can be divided into three main classes:

- Command data
- Situational awareness
- Reports

Command data is used to improve the communication of Clear and Consistent Command Intent. This traffic will generally take shape as database modifications for a *Battlefield Management System* (BMS). Here we distinguish two different types of traffic, database updates and database synchronizations. Database updates are modifications to the database, while database synchronization traffic is used to periodically synchronize the complete database contents with other nodes on the network. The majority of the traffic will therefore be update messages rather than synchronization messages. When an incoming database synchronization message is more recent than an update message, the obsolete update message can be flushed from a node's buffer. Aside from synchronization messages flushing update messages, command data should have a high delivery accuracy.

Situational awareness takes the form of for instance BFT. This traffic type deals with periodic position updates of vehicles and personnel on the battlefield. Outdated information of this type is rarely useful, therefore the network should aim for as little delay as possible when transmitting this kind of traffic. When transmitting position updates, the receiver is only interested in the most up to date version. This means that while it should have a high transmission priority, older updates can be deleted as soon as a

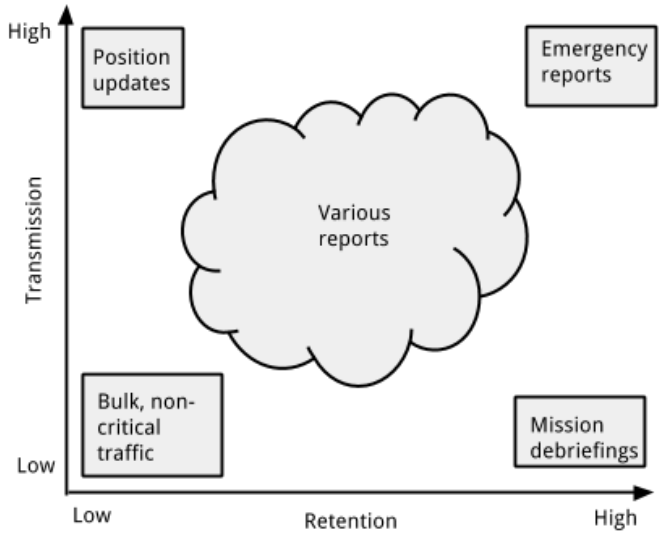


Figure 2.1: Examples of traffic types with different transmission and retention priority

newer update arrives at a node. Aside from periodic position updates, BFT traffic can also consist of for instance maps, battlefield conditions and text and imagery messages.

Last, there are the reports. These reports can be very varied and split into high- and low-priority updates. Some example of low-priority reports include a vehicle status report or mission debriefings; an example of high-priority reports would be a vehicle running low on ammunition. While a report may not have a high transmission priority, some reports should have a high delivery accuracy even if it suffers from longer delays. As the traffic is very varied, it is hard to deduce any general requirements, aside from that it should be able to be transmitted with different priorities. In addition, all of the traffic types mentioned for C2 by default limit their messages' time to live to 24 hours or shorter.

2.1.2 QoS classes

As we can see, the different traffic types on the military network impose different requirements on the network in terms of transmission priority and buffer retention. Some messages may have a high transmission priority but may easily be removed by newer messages, e.g. BFT, or messages may arrive with a higher delay but should have a high delivery accuracy, e.g. debriefings. In addition, some messages are just more important than others and should be transmitted first. For instance, a vehicle running low on ammunition should have priority over a mission debriefing. Some examples of traffic having different transmission and retention priorities can be seen in Figure 2.1. To accommodate this functionality, we propose a separate classification of transmission priority and retention class for messages.

Based on NATO defined QoS requirements for traffic on a military network, we identify the following transmission priorities, from high to low:

- Real-time
- Non real-time, but time-critical
- Non real-time, lower priority
- Best effort

As a DTN network should be delay-tolerant, it cannot offer support for real-time communications. However, we will still adapt four different transmission priorities, with Priority 4 being the highest and Priority 1 the lowest.

Rather than using retention priorities to define data retention behaviour, we define different retention classes based on the aforementioned traffic requirements. These classes determine whether messages should be stored when new messages arrive or as the transmission buffer of a network node runs out of free space. For these purposes, we define the network classes *Normal*, *Never Delete* and *Keep Most Recent*.

Normal messages behave based on classic buffer management based on transmission priorities. Once a buffer runs out of space for an incoming message, messages with a lower priority may be deleted to accommodate this message. Examples include database update messages and non-critical reports.

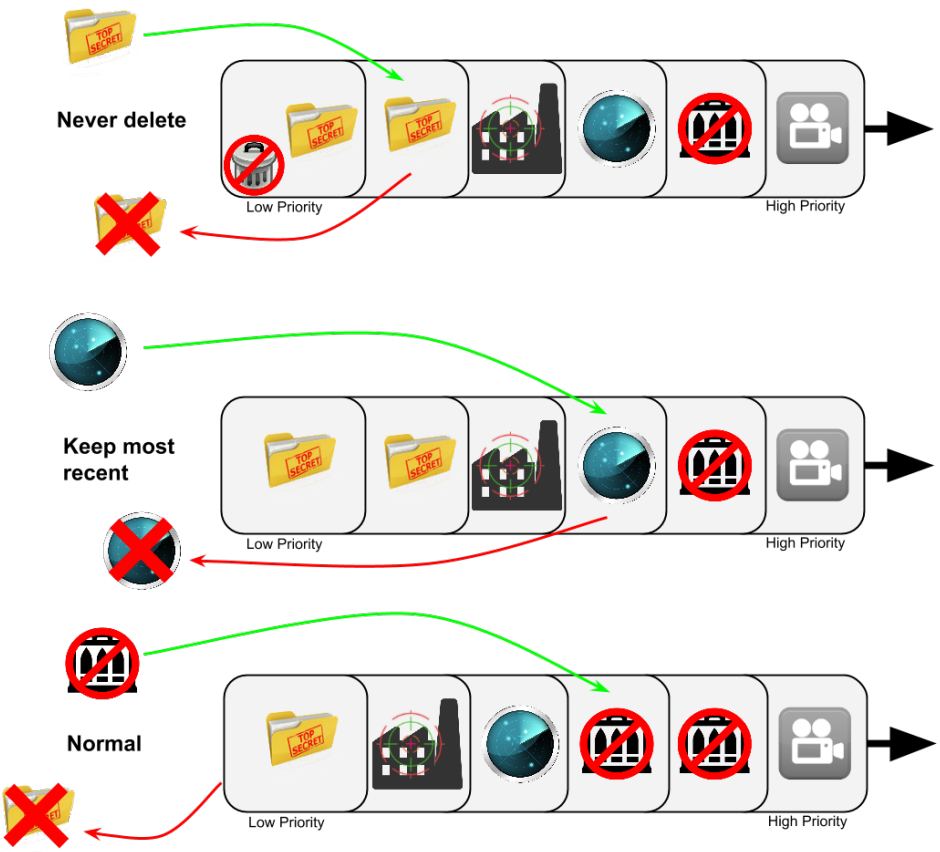


Figure 2.2: Different retention classes to be used in military DTN

Class	Deletes	Deleted by
Normal	Lower priority non-ND	Higher priority, more recent KMR, TTL expiring
Never Delete (ND)	Lower priority non-ND	TTL expiring
Keep Most Recent (KMR)	Lower priority and/or obsoleted non-ND	Higher priority, more recent KMR, TTL expiring

Table 2.1: Overview of retention classes

Never Delete messages must always remain stored within a node's buffer, regardless of their transmission priority. The purpose of these messages is that they should have a relatively high delivery accuracy, even though it may suffer some delay. When a node's buffer runs out of storage space, a higher transmission priority message may not delete messages flagged as Never Delete. One example of this type of data is a mission debriefing.

The last class is *Keep Most Recent*. These messages are allowed to delete older message types of the Normal or Keep Most Recent class, regardless of transmission priority presuming they belong to the same traffic flow. In other words, position updates for instance can only delete relevant older updates. This can occur even when the buffer still has plenty of space available. This class is introduced to limit transmitting obsolete data, optimize buffer use and overall reduce delay due to limiting the number of messages transmitted. Examples of this retention class include database synchronizations or BFT position updates. A summary of these retention classes can be seen in Figure 2.2 and Table 2.1.

The introduced retention classes do not have a direct 1-on-1 mapping to the traffic types mentioned in Section 2.1.1. Different traffic types may have a combination of different retention classes. For instance we mentioned Command data consisting of BMS traffic. When combining this BMS traffic with retention classes, a database update message may have retention class Normal, while a database synchronization message will have retention class Keep Most Recent. By analysing traffic requirements for each application operating on a military network, an exact mapping of retention classes on traffic types can be made.

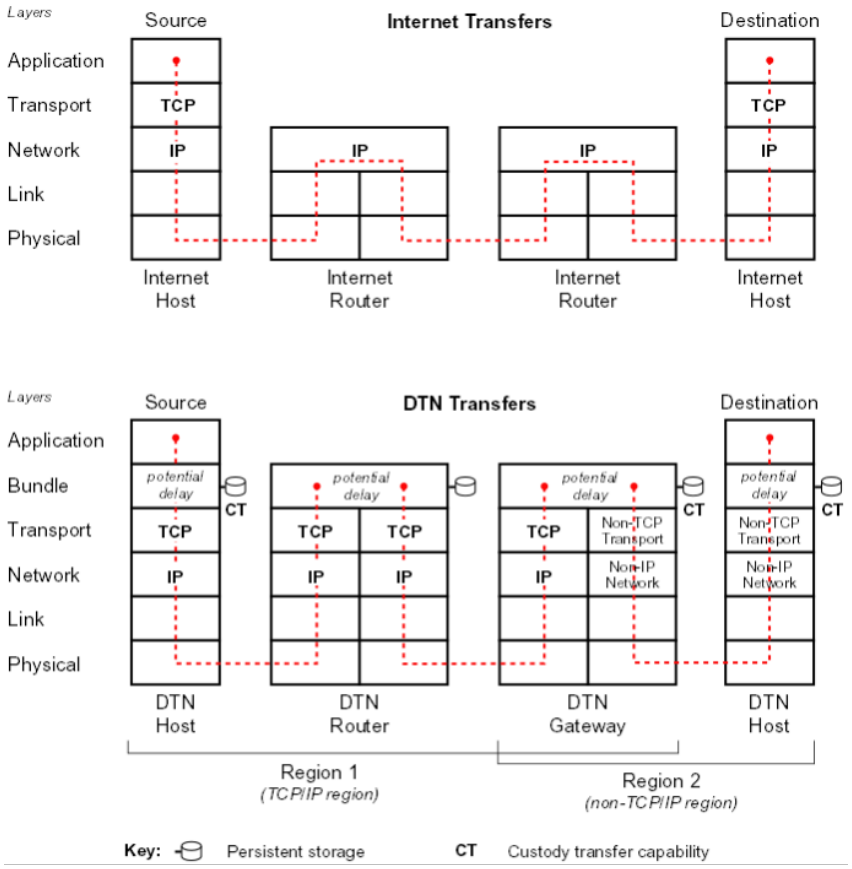


Figure 2.3: Layer structure using the bundle layer

2.2 The Bundle protocol

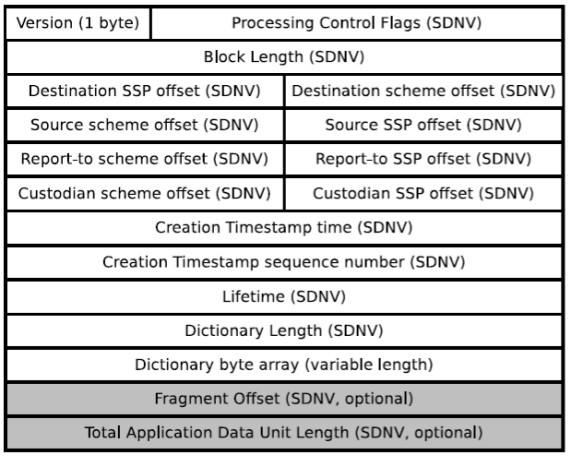
One of the core requirements in DTN is to provide SCF functionality regardless of underlying transport protocols such as for instance TCP or IEEE 802.11. To offer this functionality, a new layer is introduced just between the application layer and transport layer. This layer is the bundle protocol layer, providing DTN services to an application while managing the lower transport layers. It creates an additional overlay network to handle internode communications, tying together different lower level transmission protocols. The new model can be seen in Figure 2.3. This figure compares a classic TCP/IP Internet transfer to a Disruption Tolerant Network transfer. Note that the use of the TCP/IP stack is for illustrative purposes only and is valid for any underlying transmission layers such as UDP or non-IP networks.

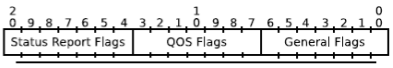
Data coming from an application is stored in so-called *bundles* at this layer. The SCF mechanism is then applied to bundles rather than individual packets. The bundle itself however, may be split into several smaller packets depending on which lower layers are used. Each intermittent hop may decide which lower level layers to use, therefore one hop may for instance communicate through TCP/IP using a wireless connection, while the next hop may communicate using a proprietary protocol. The protocol used at the bundle layer is a non-conversational protocol with simple sessions and limited or no round-trips.

The added layer provides virtual endpoints to the network. At each transmission between two nodes, the bundle layer protocol terminates the transport protocol connection, setting up a new connection for each hop a bundle needs to traverse. Because of this, retransmissions happen between two hops rather than between two endpoints as is the case with protocols such as TCP. This setup gives the bundle layer responsibility over end-to-end reliability, while hop-by-hop reliability is still handled by the underlying protocols.

To provide for retransmissions in the case of lost or corrupt data and for storage management purposes, DTN works with *custody transfers*. When a node transmits a bundle to the next DTN node with custody transfer capabilities, it will request a custody transfer for this bundle. After transmitting this custody transfer request, it will set a timer to receive a custody transfer acknowledgement from the intended custodian. Should the next hop's bundle layer accept custody, it will transmit a custody transfer acknowledgement. If no acknowledgement is received before the timer runs out, the entire bundle is retransmitted. A bundle should always stay in the custodian's storage at a node until the bundle's time-to-live (TTL) is expired or another node accepts custody.

To provide all this functionality, the bundle protocol [5] has been designed. A bundle PDU consists of one or more blocks, containing meta-data, application data or both. These blocks serve either as





Bit	Meaning
0	Bundle is a fragment
1	ADU is a status report
2	Do not fragment
3	Custody transfer is requested
4	Singleton destination
5	Application ack requested
6	Reserved
7-13	Quality of service
14	Report reception
15	Report custody acceptance
16	Report forwarding
17	Report delivery
18	Report deletion

Figure 2.4: Primary Bundle Block PDU

payload information or information normally stored in headers in lower layer PDUs. The bundle protocol however allows this information to occur anywhere within the bundle, not necessarily at the front, with exception of the Primary Bundle Block. This structure allows a certain degree of freedom in providing for different application requirements, but also different lower layer requirements. Values within these blocks are often stored as Self Delimiting Numerical Values (SDNV) as they can be variable in length.

The first block in a bundle is called the Primary Bundle Block (PBB). The PDU of this block is shown in Figure 2.4. It contains information essential for routing the bundle to its correct destination. Information stored in this block is similar to other lower layer headers, such as the bundle's source, destination and bundle processing control flags. Some other bundle protocol-specific fields are present as well however, such as the creation timestamp sequence number and dictionary fields.

Addressing within DTN using the bundle protocol is based upon nodes having one or more *Endpoint IDs* (EID) and one or more *Region IDs* (RID). RIDs are used for routing between regions, where a region is defined as a part of the network where a set of communication characteristics are homogeneous. Which characteristics should be homogeneous may vary, resulting in control of region definitions. Regions may be as large as the internet, or as small as a bluetooth piconet. Nodes acting as a gateway between different regions will therefore have multiple RIDs. Within a region, nodes are identified by their EIDs, bound only to that region. A node can have multiple EIDs to support *anycast*, *multicast* and *unicast* transmissions. These identifiers follow the same name-space syntax as the Internet's Domain Name System (DNS).

All the EIDs used when transmitting a bundle should be stored in the dictionary field of the PBB. Some fields within the block may contain similar EIDs, such as for instance the source and report-to entity. Rather than storing all addresses within these fields, fields that refer to an EID should provide an offset within the dictionary byte array where this EID can be found.

What is noticeable in the PBB is the lack of a bundle ID, common in many other transport protocols. Instead, the bundle's ID is generated by the combination of:

- Source EID
- Creation Timestamp and
- Creation Timestamp sequence number

The creation timestamp holds the real time at which the bundle was created. This timestamp, together with the lifetime denoting the TTL, determines the lifespan of a bundle. The lifetime is provided as an offset to the timestamp; should this lifetime expire, the bundle should be removed from a node's buffer/storage. The creation timestamp sequence number is a monotonically increasing value as new bundles are created. Together with the creation timestamp and source EID, it provides an unique identifier of the bundle. This use of timestamps however also results in the need of synchronized clocks among different DTN nodes. In case of unsynchronized clocks, a bundle's TTL may not be accurately handled and bundles may expire too late or too early.

Aside from this Primary Block, other special blocks may be specified as well. These blocks can provide extensions to the bundle protocol, such as security features or application specific information. These extra blocks always follow the same format: a one byte block type, block processing flags, block length and block payload. A bundle payload block will always have block type code 1. Other block types have already been reserved, however the blocktype codes 192 to 255 are free to use for private or experimental use. These so-called extension blocks are not covered in [5] and therefore nodes may not be able to process them correctly while adhering to the protocol.

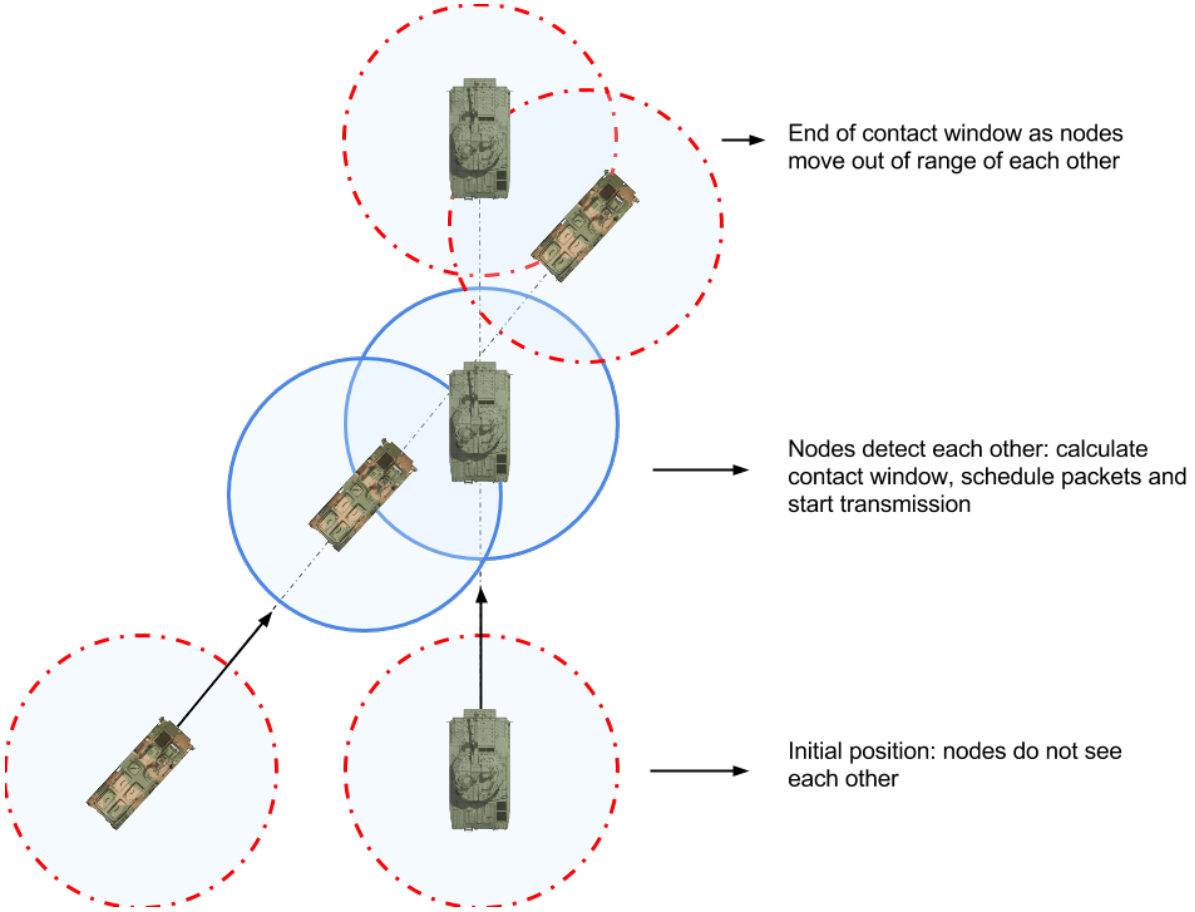


Figure 2.5: ORWAR overview

2.3 Opportunistic Routing with Window Aware Replication

Opportunistic Routing with Window Aware Replication (ORWAR) is a DTN routing protocol which attempts to utilize the limited bandwidth in DTNs in the most efficient way possible. Its secondary purpose is to optimize the transmission power and use of the limited storage in nodes. For these purposes, it makes use of message utility to make its routing decisions in a similar fashion to the RAPID[11] routing scheme. As shown in [12], there exists a message distribution method where the number of copies on the network is bound by a maximum value, depending on network structure. Copies in excess of this bound do not improve the probability of the message arriving in a significant manner, but only possibly reduce the latency. These excessive copies will therefore in general limit bandwidth efficiency. It is for this reason that ORWAR uses a similar distribution method, limiting the number of copies which are allowed to be distributed on the network.

In contrast to schemes such as PRoPHET[9], ORWAR is completely opportunistic. It does not attempt to estimate future network states of the network or in any way require higher level network topology information. The only information required by nodes is their own *location*, *velocity* and *transmission range*. The location information can be either absolute or relative to other nodes. Every bundle in a node's buffer has a utility value u_k which partially determines the transmission priority of a message. This utility value is chosen from a set of possible values, with 1 being the lowest possible utility. This utility value is then divided by the bundle's size s_k to determine the *marginal utility*. The higher the marginal utility, the higher the transmission priority of the bundle in a node's buffer. This marginal utility, in combination with the localization information, is used to transmit as much information as possible during the limited contact opportunities between nodes. This way, no resources are wasted transmitting fragmented/incomplete messages.

When a new node is discovered, ORWAR dynamically rearranges its transmission buffer, effectively separating transmission from storage. Bundles are sorted with bundles with the highest marginal utility placed at the front of the transmission buffer. Transmission in ORWAR is divided in *two phases*. First, bundles with the newly discovered node as its destination are transmitted. Only then is custody over the remaining bundles transferred to the other node, time permitting. At each transmission, the remaining transmission capacity in bytes is calculated using the estimated contact window. Should the next bundle in line to be transmitted be smaller or equal in size to this remaining capacity, it is transmitted. Any bundle with its size exceeding this value is skipped, even though it may have a high marginal utility. This way, wasted bandwidth due to incomplete transfers is limited. The bandwidth saved in this way can then

be used to transfer smaller, lower priority bundles.

Should the node's storage reach its maximum capacity, bundles will be deleted based on this marginal utility. Bundles with the lowest marginal utility are deleted first. It will keep removing bundles from the back of the storage buffer until enough space can be freed to accommodate a new bundle. However, it will only attempt to clear space as long as the removed bundles have a lower marginal utility than the incoming bundle. Should the incoming bundle's size still be too large to clear enough space in this manner, no bundles are removed and the incoming bundle is dropped instead.

When distributing bundles over the network, ORWAR makes use of Spray-and-Wait [12] mechanics. Each bundle has a limited number of copies, determined at creation by its value of u_k. Bundles with a higher value of u_k will have more copies available. At each transmission opportunity, as long as there is more than one copy of a bundle available at the transmitting node, half of the remaining copies are transmitted to the receiving node(s). If there is only one copy remaining at the transmitting node, it will only transmit this bundle if the receiving node is the destination of this bundle. This results in low latency and high delivery probability for high utility bundles, as they will have a higher transmission priority and a larger number of copies on the network.

To determine the number of copies per bundle, the parameter L is calculated first according to the method presented in [12]. This value of L determines the base number of copies per bundle. ORWAR furthermore introduces the parameter Δ which determines the offset of L for different utility values. In the ORWAR scheme proposed in [13], three utility values are used, with 1 being the lowest and 3 the highest utility. Based on experiments performed in their research, they found a value of $\Delta = L/3$ to be optimal. This however may vary for each network and is based on the number of utility values there are available. Finally, the number of copies for each utility were determined as

$$L_k \begin{cases} L + \Delta & \text{if } u_k = 3 \\ L & \text{if } u_k = 2 \\ L - \Delta & \text{if } u_k = 1 \end{cases}$$

ORWAR also supports a system for distributing acknowledgements of delivered bundles over the network. As nodes distribute copies of a bundle rather than the bundle itself, intermittent nodes will keep a copy of a bundle until an acknowledgement is received, the bundle's TTL expires, or it has to be deleted to free up storage space for bundles with a higher marginal utility. These acknowledgements are therefore used to both notify the source of successful delivery and remove copies of this bundle at intermediate custodians to prevent further propagation of a bundle that has already arrived. For this purpose, ORWAR introduces a record of *Known Delivered Messages* (KDM). This kdm_i keeps track of all bundles node i knows have been delivered to their destination. During the handshake between two nodes, these nodes exchange their respective kdm. The new information is then merged with the existing knowledge to get an up-to-date version of kdm. These records are merged in such a way that $kdm_{i_{new}} = kdm_{i_{old}} \cup kdm_j$. Bundles in a node's buffer matching any of the entries in kdm are then deleted from the buffer. In this process a node also deleted any bundle for which its TTL has expired. As soon as a bundle is transmitted to its final destination, upon confirmation of successful delivery, both the last transmitting node as well as the receiver update their kdm accordingly.

Bundles stored in the kdm are stored just as long as their TTL at the minimum. In addition, a retention time τ is introduced. This is the time a bundle should stay in kdm even after their TTL has expired. After a node's buffer has been cleared during handshaking using either the newly updated kdm or a bundle's TTL, the kdm itself is cleaned. Any bundle m in kdm for which $t > TTL_m + \tau$ is removed from that node's kdm, where t is the current time. This addition of a retention time is necessary to allow an acknowledgement some extra time to reach the original source to notify it of a bundle's delivery. The bundle itself may already have been expired and deleted at the source, though the retention time of the acknowledgement can prevent an unnecessary retransmission.

After the management of a node's buffer and kdm during handshaking, nodes will exchange the aforementioned localization information. This information is then used to estimate the contact window between them. To calculate the contact window, the information depicted in Figure 2.6 is used. Nodes exchange their coordinates (x, y), transmission range r and velocity $\vec{v_i}$. Given two nodes advancing at a vectorial speed of $\vec{v_1}$ and $\vec{v_2}$ respectively, with transmission ranges r_1 and r_2 respectively, the contact window time t_{cw} can then be calculated through

$$t_{cw} = \frac{2 * \min(r_1, r_2) * \cos\alpha}{\vec{v}} \tag{2.1}$$

where $\vec{v} = \vec{v_1} - \vec{v_2}$ and α is the angle between $\vec{v}$ and the line defined by the two nodes at contact time, depicted in Figure 2.6.

The calculation of this contact window is however non-trivial. As is explained in [17, Section 5.7], the mobility of nodes can result in inaccurate estimations of this contact window. In addition, nodes may change direction or speed during a contact window; therefore, some error margin should be taken into account when calculating this value.

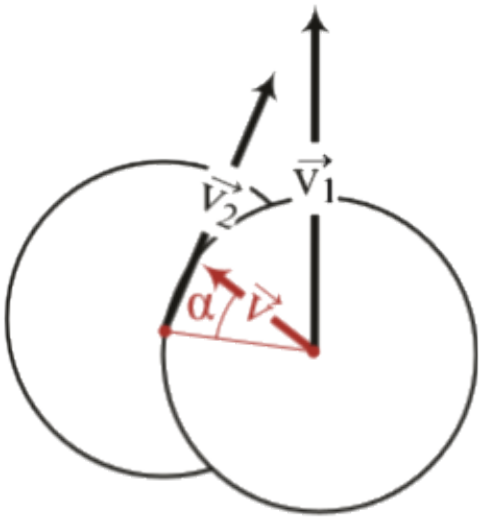


Figure 2.6: Contact window estimation in ORWAR

The calculated contact window time is then used to calculate the maximum amount of data that can be transmitted between these nodes. The maximum amount of data that can be transmitted, given the medium's transmission rate b in bytes/sec, can be calculated through

$$s_{max} = b * t_{cw} \tag{2.2}$$

During transmission, the next bundle in line, i.e. the one with the highest value of u_k/s_k, is compared to s_{max}. Should $s_k \leq s_{max}$, the bundle is transmitted. If $s_k > s_{max}$ the bundle is skipped as it cannot be transmitted in time and the next bundle in line is evaluated. At each transmission or reception of s_k a node's s_{max} is decreased with s_k. The maximum amount is decreased upon reception as well as the time spent receiving a message is time you cannot use to transmit your own. This process continues until either the end of the buffer is reached, or $s_{max} = 0$.

Upon reception, a node checks whether enough space is available for the incoming bundle. If this is not the case, it will attempt to free space as explained earlier. Should there be enough space available, a bundle is always inserted at its correct position in the buffer, determined by its marginal utility. This ensures the buffer is sorted at all times and not only after handshaking.

2.4 ORWAR Neighbour discovery

What constitutes a neighbour in DTN or not is not formally standardized and different DTN routing protocols have different descriptions of what constitutes one. As stated earlier, DTN can use underlying protocols to aid it in neighbour discovery and hop by hop routing. This means it is possible for a next-hop DTN neighbour to actually be multiple lower layer hops away. Additionally, these intermediate hops are not required to provide DTN functionality. In other words, a node connected through several IP hops to another node can be its DTN next-hop neighbour.

Due to this functionality, protocols such as OLSR and AODV can be used to determine a node's next-hop DTN neighbours as lower-layer end-to-end connectivity is always required for DTN next-hop transmissions. In addition, custom DTN-specific neighbour discovery methods using lower-layer transmission protocols have been proposed, such as [18], which can be used in combination with the epidemic [8] and PRoPHET [9] routing protocols.

However, due to ORWAR's calculation of contact windows with individual nodes, its DTN neighbours are also its direct physical neighbours. It is impossible to effectively calculate a contact window with nodes that are several lower-layer hops removed from the source. As such, existing MANET routing protocols are of little use other than for neighbour discovery of nodes within physical transmission range.

Chapter 3

ORWAR limitations & proposed improvements

While ORWAR is a very bandwidth-efficient protocol, the original research shows some limitations to the protocol which have to be overcome before its performance can be tested for a realistic mobility scenario. First and foremost, current research on ORWAR has focused on *only two nodes meeting at a time* for simplicity. The problem however, is that some features of ORWAR are not as clearly defined anymore once multiple nodes can communicate at the same time on a TDMA medium. Some examples include the contact window estimation and transmission buffer order. Aside from this problem, other issues have either been overlooked or not addressed in the currently available literature. These issues will be explained here and where possible, solutions are given.

3.1 Current limitations in ORWAR

In the available research concerning ORWAR, some issues have not been properly addressed which will reduce the performance of ORWAR when applied to a practical scenario. The following subsections will address and explain these issues and describe our proposed solutions on how to remedy the issue.

3.1.1 Unknown buffer contents of neighbours

An important, but easy to resolve, issue of ORWAR is that nodes do not know which bundles a neighbouring node already has stored. Nodes in ORWAR exchange information regarding which bundles have been delivered to the final destination, but not which bundles they are currently carrying. The problem arising from this is that nodes may transmit bundles which its neighbour already has stored in its buffer. This transmission is wasted bandwidth as no useful information is transmitted.

Even epidemic routing [8], one of the most basic routing schemes, informs neighbouring nodes of a node's buffer content. Our solution therefore is loosely based on this mechanic. Nodes already need to exchange information during handshaking with regard to their position and their kdm. We therefore also propose to add the exchange of their buffer contents to this handshaking procedure.

During the handshaking procedure, each node exchanges a collection of all bundle IDs it has stored in its buffer. This collection of IDs, which we name the Buffer Contents, or bc, is then used to prevent transmissions of bundles which a node's neighbour already has stored. The collection of bundle IDs should be transmitted after the kdm however, to keep overhead to a minimum. First, as the kdm is exchanged, both nodes have the opportunity to clear delivered bundles from their buffer. This will then reduce the size of bc before it is transmitted. Each node keeps track of all received bc_i for each node i. It will store and update this collection to keep it up to date.

When a node has multiple neighbours, the stored contents of bc_i for each neighbouring node may become out of date. This can be due to a neighbour receiving bundles from another node hidden from the local node, or the local node transmitting bundles to the neighbour. To compensate for this, whenever a node transmits a bundle, it updates the stored bc_i for each node by adding the transmitted bundle upon successful transmission. Furthermore, to compensate for bundles arriving at neighbours from unknown nodes, each node periodically retransmits its own bc to its neighbours.

In case only two nodes are communicating, bundles in local storage which are also contained within the received bc are not transmitted, as its neighbour will already have these bundles stored. When a node has multiple neighbours, the decision of which bundles to transmit based on the neighbouring nodes' buffer contents becomes non-trivial. We will further elaborate on this in Section 3.2.

3.1.2 Contact window limitations

We have already stated that the calculation of the contact window within ORWAR is a non-trivial problem. Aside from the issues concerning transmission range estimation, there are other limitations in ORWAR concerning the use of contact windows. Presuming that we can correctly estimate the contact window upon initial contact with another node, we are still left with the issue of how to use this window and of course nodes generally not travelling in a completely straight line at a constant speed all the time.

First of all, how will the contact window be used? The contact window is used to calculate the maximum number of bytes a node can transmit before the other node moves out of transmission range. This calculation however presumes that all of the calculated time can be used for transmissions. Shown in the pseudo-code presented in [19], s_{max} is only decreased upon transmitting or receiving a bundle. This window, and accompanying s_{max}, will however never be completely accurate due to for instance overhead inherent to lower level protocols. For instance, when using Wifi and its CSMA/CA multiple access mode, nodes need to take contention periods and/or RTS/CTS exchanges into account. This is time that cannot be utilized for data transfer and as such, some margin should be taken into account. This issue gets even more complex when multiple neighbouring nodes are involved, as nodes may be idle even though there are neighbouring nodes. This issue will however be elaborated in Section 3.2

The other issue is nodes changing direction or speed mid-transmission. Contact windows are only exchanged during handshaking between two nodes and not updated anymore after. Should a node change its speed or direction, the calculate contact window and hence s_{max} will be incorrect. This can never be avoided, as during travel, most nodes show minor fluctuations in speed and direction. For calculated contact windows which were short to begin with, this generally will not be too much of an issue. For large contact windows, this can prove problematic however. During a large calculated contact window, it is easy to drift into a significantly shortened, or lengthened, actual contact window. For these large windows we propose to periodically update the contact window so nodes have an actual value. These synchronisation moments should however be limited as it creates additional overhead. In addition, nodes should take into account some error margin when calculating the contact window.

3.1.3 Limited buffer maintenance

In ORWAR, the only time bundles are deleted from the buffer are during handshaking and if a bundle is successfully transmitted to its final destination. This includes bundles for which the TTL has expired. This method heavily favours many short duration contacts. In periods where a node is associated for a long time with another node, the TTL of bundles may expire. In [13], these bundles would not be deleted until the node meets another node again. This may result in bundles being transmitted while their TTL expired, wasting bandwidth, and bundles staying in the buffer longer than necessary, wasting storage. To prevent this, we propose that at each transmission opportunity, the source node first checks if the TTL of the bundle is still valid. If it is still valid, it will transmit the bundle and subtract s_k from s_{max}. However, if the TTL has expired, it will delete the bundle instead and tries to transmit the next one.

3.1.4 Limited acknowledgement support

According to the bundle protocol, when a node attempts a custody transfer to another node, the other node must reply with an acknowledgement within a certain period. After this period passes, the transfer request is timed-out and a retransmission of the bundle occurs. In ORWAR, storage space is limited and incoming bundles may not always be accepted. If the incoming bundle's marginal utility is too low and there is not enough room left in the storage buffer, the incoming bundle is dropped.

Other routing methods, such as epidemic routing, work around the issue of limited storage space by allowing the destination to request which bundles should be transmitted. ORWAR, however, does not offer any support for rejecting custody of a bundle. With no such support available, this may result in a bundle being constantly retransmitted and subsequently dropped, wasting bandwidth. We can solve this issue using two different methods.

Firstly, we can copy epidemic routing's method of allowing the destination request which bundles it receives. When implementing this method for the scenario where a node has multiple neighbours, the node will first order its transmission buffer normally. At every subsequent transmission opportunity it then determines which of its neighbours actually requested the next bundle to be transmitted. Secondly, we can introduce a system of negative-acknowledgements. If a node drops an incoming bundle as it cannot accommodate it, it sends a negative-acknowledgement. Upon receiving of this negative-acknowledgement, a node does not attempt a retransmission of this bundle and moves on to attempting to transmit the next bundle in its buffer. As bundles are ordered based on their marginal utility, the receiving node may be able to accommodate the next bundle as it may have a smaller bundle size.

3.2 Adjustments for multiple-neighbour support

The current research of ORWAR has thusfar been focussed on communication between two nodes at a time. When a node has multiple neighbours however, issues arise which have to be overcome before we can adapt ORWAR in tests involving realistic military networks. These issues include:

- Different contact windows with different neighbours
- Different neighbours having different bundles stored
- Insufficient acknowledgement propagation
- Determining which node to transmit to

The following subsections explain these issues in more detail and offer a possible solution.

3.2.1 Different contact windows

In a MANET, a node may have several other nodes within its transmission range. Based on each neighbour's localization information, all of these neighbours may have a different contact duration with the node. In ORWAR, the contact window is used to determine which bundles will be transmitted. This poses no problem when there are only two nodes associated with each other, however with multiple neighbours, the node must decide which contact window to use to order its buffer. Furthermore, nodes can not rely on their own transmitting/receiving of bundles to reduce s_{max}. In half-duplex mediums, such as the IEEE 802.11 standard, only one node is allowed to transmit at a time. In IEEE 802.11 systems, nodes use CSMA/CA to circumvent collisions and the hidden terminal problem. This however also means that sometimes nodes have to wait till the medium becomes available again, reducing their remaining contact window and s_{max}.

First, let us address the issue of determining what contact window to use. We presume that nodes are able to accurately estimate the contact window as soon as a node comes within range. Furthermore, we presume that through lower-level network discovery algorithms a node is able to detect when a node comes within range or moves out of range. Should a node meet multiple neighbours at the same time, it will choose the most favourable contact window amongst the different contact windows as its leading contact window. We propose the most favourable contact window to be the largest contact window available.

In general, only wireless communications are used in DTNs. One of the properties of wireless communications is that on a physical level, all transmissions are broadcasts and unicast does not exist. When using the largest contact window, other nodes may still listen in during every transmission. This means that even though some bundles will not be delivered to some neighbours due to them moving out of transmission range, the bandwidth is not wasted as it will still reach at least one other node. Using the largest contact window furthermore reduces the need to constantly reorder the transmission buffer, reducing overhead.

When broadcasting in such a way, the number of copies distributed on the network may increase as the number of copies remaining will be stored within the bundle. For instance, if a node has 4 remaining copies and transmits half to 3 other neighbours using a single hop broadcast, this results in $2 + (3 * 2) = 8$ copies existing on the network after the transmission. The exact impact of transmitting in this way in terms of accuracy and delay, rather than using a different algorithm to distribute copies in a single transmission, is however outside the scope of this research.

It can also occur that a new neighbour moves in range while a node already has a contact window calculated for another node. This new neighbour may have a favourable contact window compared to the current contact window. However, we do not want to reorder our transmission buffer each time a new neighbour moves into range with only a marginally favourable contact window. This may lead to a node constantly reordering its buffer if it frequently meets new neighbours rather than transmitting. Also we should remember that each contact window should have some error margin and therefore a marginally better contact window might not matter in the end.

To resolve this issue, we introduce a contact window threshold θ_{cw} (in %). When a node already has calculated a contact window with another node i, t_{cw_i}, the new neighbour must produce a contact window which exceeds this by θ_{cw}. In other words, if a node is communicating with node i and another node j arrives, the new contact window is given by the following equation:

$$t_{cw} \begin{cases} t_{cw_j} & \text{if } t_{cw_j} \geq t_{cw_i} + \theta_{cw} t_{cw_i} \\ t_{cw_i} & \text{otherwise} \end{cases} \tag{3.1}$$

If this results in a new, larger, contact window, the transmission buffer should be reordered. Otherwise, nothing happens. Setting this threshold too low will result in a node reordering its buffer too often. Setting it too high however, may lead to inefficient propagation of bundles.

To resolve the issue of a shared medium, we introduce an *effective contact window*, t_{eff}. To solve this issue, we presume that each node gets a fair share of the medium through lower layer channel access mechanisms. This means that for n neighbours, each node will get $1/(n+1)$ share of the medium's resources. We therefore define the effective contact window of a node with n neighbours as:

$$t_{eff} = \frac{t_{cw}}{n+1} \tag{3.2}$$

In the original research, s_{max} was decreased as a node transmits or receives bundles. The effective contact window however estimates how many bundles can be transmitted by this node on the medium. We therefore redefine s_{max} as:

$$s_{max} = t_{eff} * b \tag{3.3}$$

Using this new value of s_{max} we now only decrease it when a node transmits data. This should provide a work-around for time spent idle waiting for the medium to become available.

This also means that as soon as a node loses contact with another node, t_{eff} should be recalculated using an up to date number of neighbours. When comparing contact windows to determine the most favourable window, t_{eff} should be used rather than t_{cw}. As soon as a node leaves however, even though t_{eff} might improve by more than θ_{cw}, the buffer should never be reordered. Only s_{max} should be recalculated in this scenario as the increased contact window is due to less contention for network resources rather than meeting a new neighbour with favourable mobility.

3.2.2 Different buffer contents

Each node that is discovered on the network will have different bundles stored in its buffer. We already discussed the issue where nodes need to know which bundles its neighbours have stored to prevent duplicate transmissions. When only two nodes are communicating, the issue becomes trivial as nodes should refrain from transmitting bundles which the other node already has stored. However, as a node meets different neighbours, some of its neighbours may already have received the bundle while others have not. This creates a problem where nodes need to determine which bundles to transmit.

The focus of ORWAR is to maintain bandwidth efficiency, therefore we do not want to waste too much bandwidth on bundles which are already widely spread. For this purpose we have added another threshold θ_{bc} (in %), which is defined as the percentage of neighbouring nodes which are allowed to already have a copy of this bundle stored before transmission. For a bundle b_k, let S denote the set of neighbours n_i with associated buffer contents bc_i, defined by

$$S = \{n_i | b_k \in bc_i\} \tag{3.4}$$

The percentage of neighbours Φ_{b_k} already having a copy of b_k, for N neighbours, is therefore given by

$$\Phi_{b_k} = \frac{|S|}{N} \tag{3.5}$$

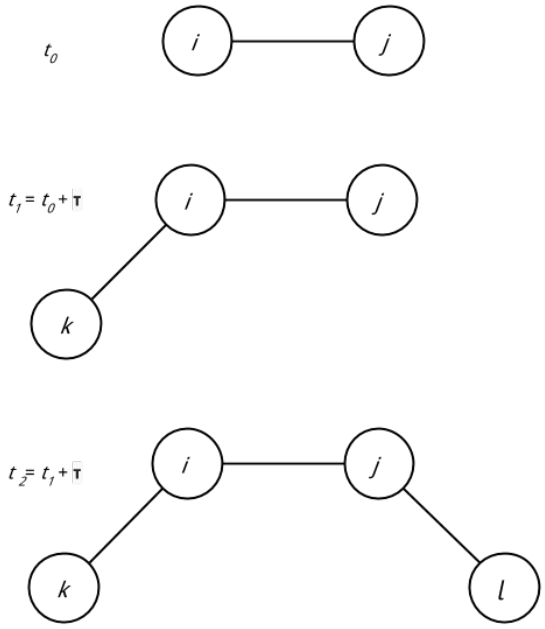
When reordering the transmission buffer, only bundles for which $\Phi_{b_k} \leq \theta_{bc}$ should be included. Regardless of Φ_{b_k} however, bundles which have one of the neighbours as its destination should always be included in the transmission buffer. Choosing θ_{bc} too strict may prevent bundles from being propagated over the network, whereas choosing it too lenient may lead to wasted bandwidth.

Using this threshold may not produce a transmission buffer for which the total size $s_{buffer} \geq s_{max}$. In this case there are two options, the node leaves the buffer as it is, or it can decide to adjust the threshold to be more lenient until $s_{buffer} \geq s_{max}$. Not adjusting the threshold may free up network resources for its neighbours to transmit barely propagated bundles. However, this also comes at the risk of stunting the propagation of its own bundles.

3.2.3 Insufficient acknowledgement propagation

The kdm collections are the only way in ORWAR to propagate acknowledgements over the network. When only two nodes are connected to each other, the kdm of each node will always be up to date as it is exchanged during handshaking and updated when a bundle has been successfully transmitted to its destination. When a node has multiple neighbours however, the kdm of different nodes may get out of sync. Presume there is a node j which moves into range of another node i. At some later moment, node k also moves within range of i, but not within range of j, see Figure 3.1. In this scenario, after handshaking, $kdm_i = kdm_k$, but there is no guarantee $kdm_j = kdm_k$.

Furthermore, node j may also meet another node l which is only within range of node j. Presume that now node j transmits a bundle to node l, where l is the bundle's destination. Now both nodes j and l add that bundle to their respective kdm, but nodes i and k do not receive this acknowledgement.



		t_0	t_1	t_2
i	kdm_i		{D}	{D}
	bc_i	{A,B}	{A,B,C}	{A,B,C}
j	kdm_j			{C}
	bc_j	{C,D}	{A,B,C,D}	{A,B,D}
k	kdm_k	{D}	{D}	{D}
	bc_k			{A,B,C}
l	kdm_l			{C}
	bc_l	{E}	{E}	{E}

Figure 3.1: Network setup where each node has a different kdm

When the kdm of different nodes get out of sync, nodes may start transmitting bundles which have already arrived at their destination. To prevent this from happening, we need to improve the way acknowledgements are handled within ORWAR. One improvement we propose is nodes checking incoming bundles whether they are already listed in their own kdm. Should an incoming bundle already be listed in the node's kdm, it will transmit an acknowledgement to the transmitter so it can also add this bundle to its kdm and clear all remaining copies from its storage.

Should the transmitting node have received this bundle through a custody transfer by another node in its range, there is only a low risk that this other node retransmits this bundle. As a node transfers a bundle to node i, it adds this bundle to bc_i, therefore stopping it from retransmitting the bundle to this node. It may still attempt a retransmission if it meets new nodes and may therefore reorder its transmission buffer. When reordering its buffer, it will again evaluate this bundle using θ_{bc}. To prevent further propagations of an already delivered, we therefore also propose that as soon as a bundle reorders its transmission buffer, it requests an up to date kdm from its neighbours.

Furthermore, we make use of the broadcasting nature of wireless transmissions by sending acknowledgements as a single-hop broadcast. This way, for each bundle that has been delivered, or erroneously transmitted to a custodian which already has it stored in its buffer, multiple nodes may be able to update their kdm without using more network resources. We limit this to only a single-hop broadcast to prevent flooding the network with acknowledgements, which would waste too much bandwidth.

3.2.4 Determining next-hop node

When we assume that only two nodes will meet each other at a time, bundles are always transmitted in sequence based on their marginal utility to the other node. Of course, this presumes the other node does not already have the bundle and the bundle fits within the remaining contact window. However, as soon as a node has multiple neighbours, it has to decide who to transmit a bundle copy to.

As ORWAR is opportunistic, with one of its advantages being not requiring topology information, it is hard to determine which bundle should be transmitted to which neighbour. As it stands, the only decision mechanic we have to select which node a bundle is transmitted to is whether or not that node already has a copy of the bundle. Furthermore, when we transmit a bundle to another node, we need to decide whether or not this bundle will also be transmitted to more neighbours. Unlike traditional ad-hoc routing protocols such as OLSR, the next-hop of a bundle is not clearly defined.

Determining the next hop of a bundle has impact on both the delivery ratio of this bundle but also on that of other bundles stored at the same node. Choosing the wrong next hop of a bundle will decrease the probability of the bundle arriving at the destination. Furthermore, if a bundle needs to be transmitted to multiple neighbours using several unicasts, this may waste a portion of the contact window. This in turn limits the number of bundles which will be transmitted, possibly reducing the delivery ratio of bundles with a lower marginal utility.

To resolve this issue, we make use of the wireless channel characteristic that all transmissions by a node can be sensed by all of its neighbours. Instead of unicasting bundles to custodians, we use a

one-hop broadcast to transmit the bundle to a node's neighbours. This way, only a single use of the medium is required to propagate copies of the bundle to other nodes. We limit unicasts to bundles being transmitted to their final destination. In this case, we do not want other nodes to accept custody over the bundle as this will add unneeded copies of a bundle to the network.

Chapter 4

Adaptations for QoS provisioning

4.1 Bundle protocol

The bundle protocol as specified in [5] already offers some QoS support as the primary bundle block contains QoS flags. Referring to Figure 2.4, the primary bundle block has 7 bits reserved for QoS. In our proposed QoS mechanism for military networks, we make use of 4 transmission priorities and 3 retention classes. This totals to 12 different combinations of transmission and retention. These 12 combinations are easily contained within the 7 bits available in the primary bundle block. We therefore do not need to make changes to the bundle protocol to identify the transmission priority and retention class of a bundle.

Using ORWAR and our retention classes introduces the need to transmit additional information, not contained within the primary bundle block. Therefore, to provide support for ORWAR and our retention classes, we need to add a new bundle header block. According to [5], block type codes 192 to 255 are free to use for experimental or private use. We should therefore use any of these unused codes to define our ORWAR-specific header block.

First, ORWAR makes use of Spray-and-Wait mechanics to distribute its bundles over the network. This limits the number of copies of a bundle available on the network. In addition, the number of copies remaining determines whether or not this bundle should be transmitted by a node. It is wasteful of bandwidth and storage capacity to physically store multiple copies of a bundle. Instead, it is more efficient to keep track of a counter in each bundle copy tracking the number of copies remaining for this bundle. At each successful transmission this counter is halved until only 1 copy remains at the current node. This counter should be added to our custom header block.

Second, when using the Keep Most Recent retention class, bundles need to be able to filter which bundles can be deleted by a more recent bundle. Merely replacing bundles originating from the same source or directed at the same destination for instance, results in unwanted behaviour. Based on the fields available in the primary bundle block, bundles from unrelated applications are able to delete each other. For instance, by filtering on either source or destination EIDs, bundles containing debriefing information can delete bundles transmitting position updates.

To counter this issue, we add a field called *stream ID* to the custom header block. This stream ID is determined by an application to divide its traffic into separate “streams”. Different applications will have different stream IDs, however within an application different stream IDs may also be defined. This means that a database application and a targeting system will use different stream IDs. However, the targeting system itself may use different stream IDs to distinguish between visual and positioning data. When a Keep Most Recent bundle is received by a node, it is only allowed to remove bundles which share its stream ID.

4.2 ORWAR

Due to the marginal utility in ORWAR, priority scheduling based on transmission priority is already present. By replacing the utility value with transmission priority, bundles will be sorted based on the highest transmission priority per bit. As we replace the utility value with our transmission priorities, we do not need to change anything within ORWAR’s transmission algorithm. This means Priority 4 messages will have highest priority and Priority 1 the lowest priority. Our added retention classes have no influence on the way bundles are transmitted.

Our added retention classes, however do influence which bundles to keep and which to remove when a bundle is received. Therefore, we need to adapt ORWAR’s receiving algorithm. The current algorithm either stores a bundle at the correct location within the buffer if there is enough storage space available, or first tries to free enough space if there is not enough space available. If a bundle b_k is received with size s_k and utility u_k, yet there is not enough space left to store it, it will attempt to delete

bundles with a lower marginal utility from its buffer B. This clearing algorithm works as such: if $\exists b_n \in B$ such that $(u_k/s_k > u_n/s_n) \wedge (s_k \leq \sum_{i=n}^{last} s_i)$, delete $b_n, \ldots, b_{last}$. If not, drop the incoming bundle.

This algorithm needs to be expanded to provide support for Never Delete and Keep Most Recent bundles. First of all, when calculating if enough space can be freed through $\sum_{i=n}^{last} s_i)$, the node should not take into account bundles which are classed as Never Delete. Things get a little bit more complicated when introducing Keep Most Recent bundles, however. We do not want to delete bundles if no new bundles will be inserted in their stead. When a Keep Most Recent bundle is received, the receive algorithm is split in two parts.

First, we check which older bundles will be deleted should we accept the bundle. This results in a set of obsolete bundles O, defined as

$$O = \{b_n \mid (sid_n = sid_k) \wedge (t_{c_n} < t_{c_k}) \wedge (class_n \neq never\_delete)\}$$

with sid being the bundle's stream ID and t_c its creation time. Using this set, we can calculate how much space would be freed by deleting these obsoleted bundles, namely $s_O = \sum_{n \in O} s_n$. If there is already enough space to accept the incoming bundles, we remove all the bundles contained in O from B and accept the bundle. Similarly, if $s_O \geq s_k$, we also remove all bundles contained in O and accept the bundle.

Should it still be impossible to clear enough space this way, we move on to the second part. This part is similar to the original ORWAR algorithm, with a few minor differences. When we check whether any bundles with a lower marginal utility can be deleted, we do not take into account bundles already present in O as these will be deleted anyway due to obsolescence. This will prevent us from calculating a wrong value for the possibly freed space if a bundle is both obsolete *and* has a lower marginal utility. If $\exists b_n \in B$ such that $(u_k/s_k > u_n/s_n) \wedge (class_n \neq never\_delete) \wedge (s_k \leq \sum_{i=n}^{last} s_i + \sum_{b_j \in O} s_j)$, we delete the contents of O as well as $b_n, \ldots, b_{last}$ from the buffer and accept the bundle. If not, there is no way we can accommodate the incoming bundle and we drop it. Note that as we drop the incoming bundle, the contents of O stay in the buffer.

One final adaptation needs to be made to the receiving algorithm to accommodate Keep Most Recent bundles. In DTN, bundles are propagated by transmitting several copies over the network. This means that while a bundle may be removed from a node as it has become obsolete, copies of this bundle may still exist somewhere else on the network. This means that as bundles are removed by Keep Most Recent bundles, these same bundles may be received by the node at some later time. To prevent this from happening, each node should record the creation time t_c of the most recent accepted Keep Most Recent bundle *for any given stream ID*. When a node receives a new bundle k, where $class_k \neq never\_delete$, it compares t_{c_k} to the t_c stored for sid_k if its available. Only when $t_{c_k} \geq t_{c_{sid}}$ will it consider accepting the bundle. If $t_{c_k} < t_{c_{sid}}$, it will reject the bundle outright. This $t_{c_{sid}}$ value should however be linked to the TTL of the Keep Most Recent bundle to prevent endless storing of these values. Should the TTL of the original Keep Most Recent bundle expire, then $t_{c_{sid}}$ can also be cleared from memory.

Chapter 5

Measurement setup & Simulation approach

5.1 Mobility model

The proper choice of a mobility model greatly influences the perceived performance of a protocol during simulation. To performance basic tests on our protocol to analyse its behaviour, we make use of several different models as displayed in Figures 5.1 and 5.2. Figure 5.1 displays three different simplified mobility models we use to analyse the behaviour of ORWAR and our added retention classes. In each of these simplified models, we use three nodes: `A`, `B` and `C`, which behave differently based on the model used.

First we have a model where all three nodes are stationary. Nodes `A` and `C` are invisible to each other, with `B` being visible to both. This effectively creates a permanent end-to-end connection between `A` and `C`. From this point onwards we shall refer to this model as the *stationary* model.

The second model is a model where the middle node, `B`, is moving in a linear "back-and-forth" pattern between nodes `A` and `C`. Again, nodes `A` and `C` are invisible to each other and rely on node `B` to relay their messages. In this case, intermittent connectivity is introduced as node `B` intermittently moves out of range from either node `A` or `C`. This mobility shall be referred to as the *linear mobility* model.

The last model is similar to the second. However, in this model node `B` moves in a different pattern between the nodes. Rather than simply going back and forth, it now moves in an eight-shaped pattern, changing direction within the transmission range of either node `A` or `C`. We shall refer to this model as the *non-linear mobility* model.

For the non-stationary models, we assume `B` to move with a constant speed. In addition, all nodes have an equal transmission range of 25 meters and node `B` moves to within a distance of 5 meters away from nodes `A` and `C`. To provide a fair comparison between the linear and non-linear mobility models, we make sure that the contact window between `B` and either `A` or `C` is equal. For this purpose, we have chosen a speed of 5 m/s in the linear mobility model and $5\sqrt{2}$ m/s in the non-linear mobility model. This results in a contact window of 8 seconds for each node, presuming node `B` does not change its speed when turning or reversing. Also note that in these models, there is a brief moment of end-to-end connectivity as node `B` crosses over from one node's transmission range into the other.

Figure 5.2 shows a mobility model, developed in [20]. This mobility model is a more realistic model of a group of military vehicles. Military vehicles rarely move in a completely random manner, therefore the mobility model should reflect this. This model consists of four vehicles moving together as a formation using a random waypoint model. These vehicles do not always stay in an identical position relative to each other however. As the vehicles move, the actual distance between them may vary over time as they for instance drive around obstacles. Their formation should however remain relatively unchanged.

In this model, the vehicles' formation is determined by the distance between vehicles at their initial position. Each individual vehicle however also has an individual area around it in which it is allowed to deviate from its initial position. In this area vehicles use a random waypoint model. Due to their individual movement, they may move in and out of each other's transmission range. In our model, in their initial position, the distance to the nearest node is 25 meters. We therefore set the transmission range to match this distance so there is an end-to-end connection in their initial position. To simulate this model, we make use of traces generated in [20].

In [21] and [22], other mobility models attempting to simulate real military networks have been analysed. Both researches have confirmed that mobility in military networks is never completely random and there is a strong group hierarchy in the involved nodes. In addition, both researches describe the use of a grouped random waypoint model, where nodes move as a group but with random variations within the group. These researches therefore serve as validation for the use of the model described in [20]

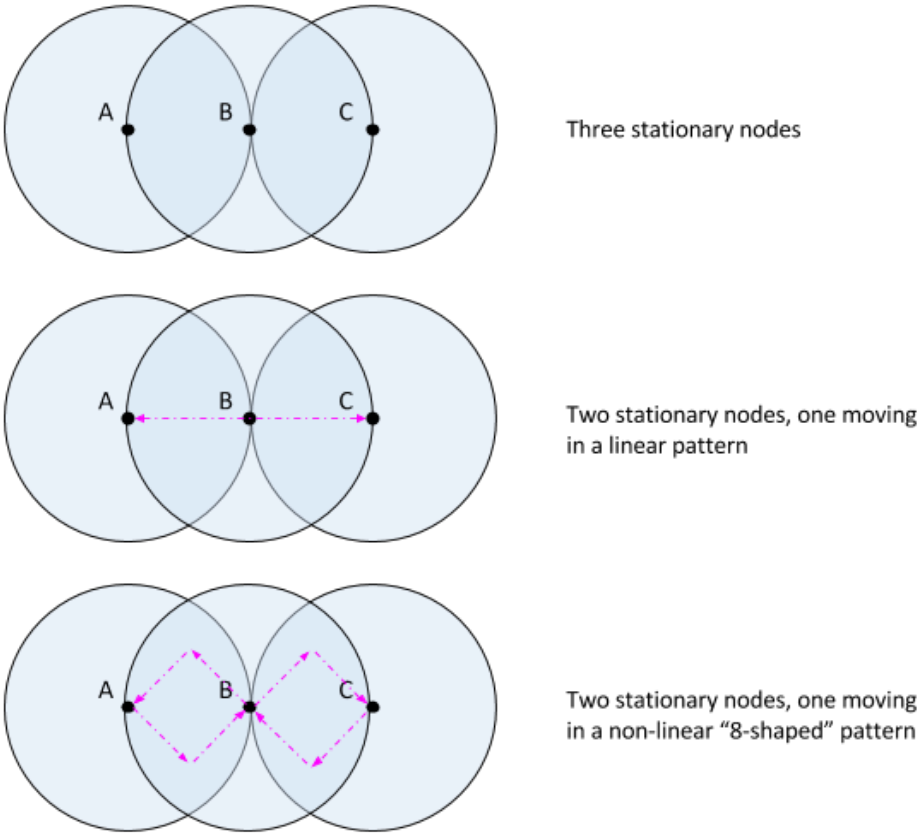


Figure 5.1: Simplified mobility models

5.2 ORWAR & Network setup

In our simulations we use the ORWAR protocol adjusted for the support of multiple neighbours, as described in Section 3.2. In our simulated version we increase θ_{bc} if a transmission buffer can not be completely filled with bundles using the current value. This decision was made as not to risk reduced propagation of bundles.

Our version of ORWAR was implemented in `C++`, using the pseudo-code derived from [19]. Furthermore, we make use of an abstract model of a network through the use of a custom simulator. This simulator simulates a TDMA environment with a strict scheduler which determines which node can transmit. This scheduler ensures each node gets a fair share of the network resources. We use abstract representations of bundles in our simulations, choosing not to concern ourselves with the underlying layers' transport mechanisms. As bundles are abstract in our simulations, this also means we do not implement the complete bundle protocol and the extra bundle headers. We have chosen this approach as we are only interested in the behaviour of ORWAR when we introduce QoS features.

Furthermore, in our simulations, we presume faultless transmission of bundles between nodes, handled by the lower layers. In addition, we do not track the behaviour of individual bundles. This means that even though a bundle is rejected by the receiving node, we do not concern ourselves with retransmission. This situation is treated as if the transmitting node received a negative-acknowledgement as described in Section 3.1.4.

The properties of the simulated network are shown in Table 5.1.

5.3 Approach

Our experiments are split into two parts where we first analyse the behaviour of ORWAR where we gradually introduce transmission priorities and retention classes. In this analysis, we make use of the simplified mobility models described earlier. The second part of our experiment deals with comparing the performance of ORWAR with and without our QoS provisioning. For this purpose, we make use of the realistic model as presented in Figure 5.2.

When analysing the behaviour of ORWAR, we insert bundles using node `A` as the source and node `C` as the destination. This approach is chosen to establish an indication of how the different variables are influencing each other in a single traffic flow, with a single source buffer. Using this insertion method, we generate a bundle stream and measure the loss and delay of these bundles.

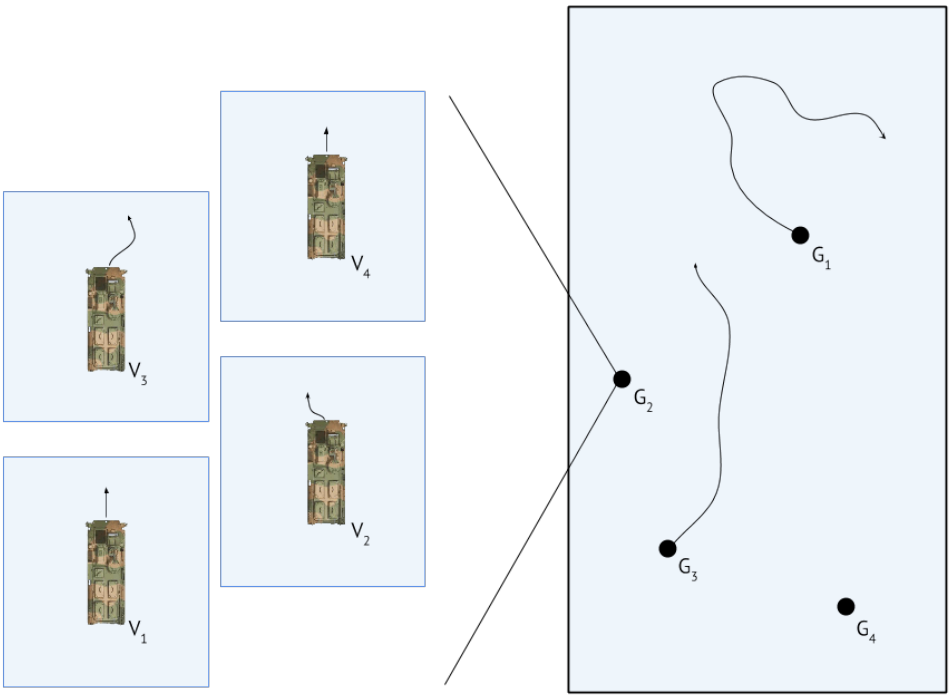


Figure 5.2: Realistic mobility model for a military MANET

Buffersize	500 bytes
Bundlesize	5 bytes or 1-10 bytes
Medium	half-duplex TDMA, 100 bytes/sec
TTL	20 seconds
Stream IDs	16 total, of which 4 can contain Keep Most Recent bundles
Measurement time	300 seconds with topology updates every second (retention class-only behaviour) 3600 seconds with topology updates every second
θ_{cw}	20%
θ_{bc}	50% with possible 10% increments up to a maximum of 90%

Table 5.1: Simulated network properties

Here we distinguish two types of loss. First there is the *network loss*, defined as the loss of bundles *after* they have been injected onto the network. This loss can occur by either the TTL expiring, or if all copies on the network are deleted by higher-priority bundles. Note that this loss is not calculated for individual copies, but for the availability of a certain bundle on the network. In other words, as long as at least one copy of a bundle is still available on the network, it is not deemed lost.

The other type of loss is the *total loss*, which encompasses both the network loss and bundles lost *before* being injected on the network. As a node stores copies of bundles, the storage buffer will fill up over time. The situation may arise where it cannot accommodate an incoming bundle, either from another node or from an application, and it will be dropped. Incoming bundles from other nodes that are dropped will be part of the network loss, presuming all other existing copies are lost as well. This results in the total loss being a sum of the network loss and bundles which are not injected in the source buffer.

To state these losses more formally, let a denote the number of arrived unique bundles, c the number of bundles created by applications, b_i the number of bundles injected on the network and b_r the number of bundles created by applications but rejected at the source buffer. We then obtain the loss L through

$$\begin{aligned} L_{network} &= \frac{a}{b_i} \\ L_{total} &= \frac{a}{c} = \frac{a}{b_i + b_r} \end{aligned} \tag{5.1}$$

We first test our added retention classes using the simplified mobility models by generating bundles of equal transmission priority and equal bundlesize, as shown in Table 5.1. In this situation, only the retention priority and stream ID influence the network's behaviour as buffersize, transmission range and bitrate are kept equal. Note that this removes some of the functionality of standard ORWAR as now there is no scheduling of bundles based on the utility per bit. Bundles in this case will be transmitted in a FIFO manner.

Using this setup we measure the delay and loss for varying ratios of retention classes. To determine the individual influences of each retention class, we perform measurements with only two retention classes at a time.

In our experiments we provide a stream of bundles at the source node, which in reality will be generated by varying applications. By changing the creation rate of bundles, we simulate different loads on the network. In our first experiments we create a different fixed numbers of bundles per second to analyse the network's performance. This allows for easier validation of results as the injection rate is constant. We then repeat these simulations using a Poisson distributed creation rate with different mean rates c to provide a more realistic view of the network.

These simulations will provide a good understanding how the retention classes influence each other. We continue to run the same experiments, still keeping the bundle size constant but adding the transmission priorities. Transmission priorities are assigned independent of retention priority and uniformly distributed. By assigning transmission priorities, priority scheduling occurs in the nodes and bundles may be dropped in favour of higher priority bundles. However, as size is kept equal, high transmission priority bundles will always be transmitted before lower priorities.

For our last experiments to determine the behaviour of ORWAR, we will also vary the bundlesize of individual bundles. In this scenario, if all the transmitted bundles have the Normal retention class, its behaviour will simulate the non-QoS provisioned ORWAR. This is the final analysis we perform on the network behaviour of ORWAR and our proposed QoS provisioning.

After our analysis of ORWAR's behaviour, we switch to using the realistic mobility model. Using the results gathered from our analysis using the simplified models, we attempt to determine some optimal parameters for our QoS-provisioning in terms of retention class distribution. We now do not limit ourselves anymore to a single source and destination; every node on the network will now generate bundles, randomly selecting a destination on the network. Every node will generate bundles based on a Poisson distribution with a mean creation rate c, which will be varied to depict different network loads. At the end, we measure the $L_{network}$ and L_{total} on the network, together with the mean delay. These values are then used to compare the performance of normal ORWAR to our version with added QoS-provisioning.

Chapter 6

Results using simplified mobility models

6.1 Retention classes only

To determine the influence of the added retention classes, mentioned in Section 2.1.2 we first test our version of the ORWAR protocol with the transmission priorities removed. Additionally, all bundle sizes are equal. In this case, the utility per bit is equal for all created bundles and hence there is no priority scheduling based on transmission priority. The goal of these experiments is to determine the influence of our retention classes on the network loss and delay. Note that this does not accurately reflect ORWAR as a whole, as the use of bundle size and transmission priorities is a big factor in its performance. However, it does present us with some insight into the behaviour when using retention classes.

6.1.1 Stationary model

Basic ORWAR performance

As a baseline comparison, we first analyse the accuracy of normal ORWAR using this setup and a constant, deterministic, injection rate. The results for this setup, using the stationary mobility model, are shown in Figure 6.1.

What is already significant in this figure, is that at a creation rate larger than 4 bundles per second, losses start to occur. As the total loss increases, but the network loss does not, we can determine this loss is due to bundles not being injected on the network. Further analysis of this behaviour shows that this behaviour occurs due to ORWAR performing poorly with stationary nodes.

This poor performance is caused in the way acknowledgements are propagated within ORWAR. In ORWAR, these are exchanged from node to node using the KDM-vector. However, this vector is only exchanged during handshaking. The result of this is that in the stationary model, acknowledgements

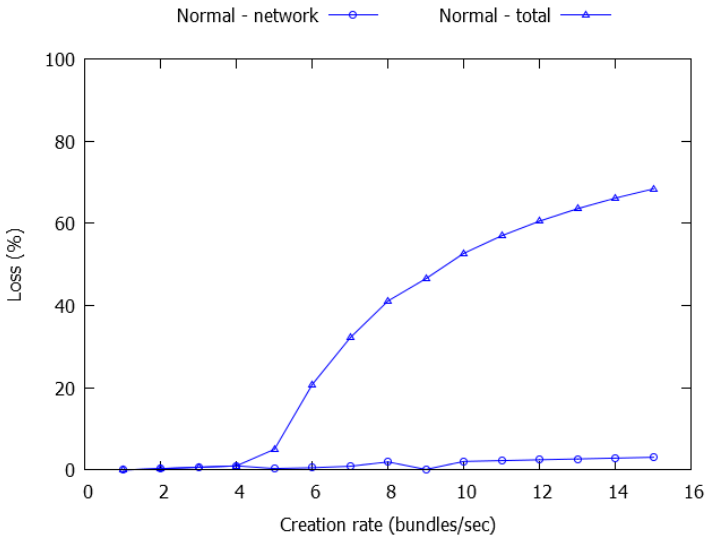


Figure 6.1: Loss in normal ORWAR without transmission priorities and constant bundle creation rate for the stationary mobility model

do not reach the source node as there is no handshaking aside from the initial setup. This leads to the source buffer filling up, as now only the expiry time of bundles is able to provide the means to clear bundles from the buffer. Therefore, new bundles are unable to be injected onto the network as bundles that are already delivered are still in the source's transmission buffer, increasing loss.

The intermediary node, however, *will* receive an acknowledgement and remove the transmitted bundle from its buffer, due to it being the final hop and receiving an acknowledgement directly from the destination. This explains why no loss occurs on the network itself, as bundles injected on the network still will be able to reach the destination because the intermediate node still has a clean buffer.

Note that this issue does not only occur with stationary nodes. It can occur as long as there is more than one hop between a source and destination, with long periods of connectivity between these nodes. In these situations, the acknowledgements are not properly propagated back to the source.

The point at which this problem occurs is based on the TTL and source buffer size. Once the creation rate has a value where the source buffer will reach maximum capacity if bundles are only deleted when they expire, loss starts to occur due to non-injection. More formally, loss starts to occur when

$$c * TTL \geq s_b \tag{6.1}$$

where c is the bundle creation rate and s_b the buffer size in bundles. In our simulations, the TTL is 20 seconds and the buffer size is 500 bytes. As all bundles have an equal size of 5 bytes, this means it can accommodate 100 bundles. When bundles are only deleted through TTL expiration, loss will therefore start to occur when the bundle creation rate $c \geq 5$, which we see in Figure 6.1

The acknowledgement propagation improvements mentioned in Section 3.2.3 do not remedy this issue. In the mentioned scenarios, the intermediate node will not receive another transmission of the delivered bundle as the source node knows the intermediate node already received that bundle. Therefore, this calls for another improvement to the manner in which acknowledgements are propagated on the network.

To remedy this issue, we introduce a periodic update of the KDM-vector at each node. In our experiment, we force a KDM-vector synchronization between nodes every second. This solution is only implemented in the static mobility model, as in the other models this issue does not occur. By forcing a synchronization of these vectors, we improve buffer efficiency and in turn reduce the total network loss. The cost of this is a tiny increase in bandwidth use. As this KDM-vector is nothing more than a vector of bundle IDs and associated expiry times, the bandwidth required is very small compared to actual bundles.

Implementing these changes, we obtain the results shown in Figure 6.2. Here we can clearly see the effects of having an end-to-end connection available between source and destination. Our medium has a bit rate of 100 bytes/sec; bundles sent in this simulation all have the same size of 5 bytes, resulting in a channel capacity of 20 bundles/sec. All nodes also share the same medium, even though node `A` is hidden from node `C`, see Figure 5.1. We also presume that in a real world scenario, each node receives a fair share of the medium. Therefore, we conclude that each hop in the stationary model has a capacity of 10 bundles/sec.

This means that, presuming there is an end-to-end connection, 10 bundles/sec can be transmitted from source to destination every timeframe. Hence, as can be seen in Figure 6.2, any creation rate of $\leq$10 bundles/sec will have 0% loss as created bundles will immediately be transmitted to the destination within the same timeframe. Beyond this point, every second more bundles are injected on the network than can be transmitted. This results in the buffer gradually filling over time until it is full and all bundles in excess of 10 each second are rejected by the buffer. More formally, in this mobility model, the total loss is given by:

$$L_{total} = \begin{cases} 0 & \text{if } c \leq r_b \\ \frac{c - r_b}{c} & \text{otherwise} \end{cases} \tag{6.2}$$

where c is the creation rate in bundles/sec, r_b the network's per-hop transmission rate in bundles/sec and L_{total} the network's total loss percentage.

Likewise, due to the capacity, a limited number of bundles can be transmitted during each experiment, regardless of creation rate. Our buffersize is 500 bytes, which is equal to 100 bundles using the current setup. At the end of the simulation, there will still be 90 bundles remaining in the buffer at the end of the simulation, presuming the creation rate is larger than the transmission rate. This remainder is the only network loss, as these are the only bundles that will not reach their destination. The network loss can therefore be given by

$$L_{network} = \frac{s_b - r_b}{t * r_b}, \text{ if } c > r_b \tag{6.3}$$

with $L_{network}$ being the network loss percentage, s_b the buffer size in bundles, r_b the per-hop transmission rate in bundles/sec, t the total simulation time in seconds and c the creation rate in bundles/sec. In our experiments, this results in a constant network loss of approximately 3% once $c > 10$. However, it is easily shown that for $t \to \infty$, this loss becomes 0%. As we add extra variables such as transmission priorities and varying bundlesizes to our experiments and increase the simulation time later in this section, this loss due to left-over bundles after simulation becomes negligible.

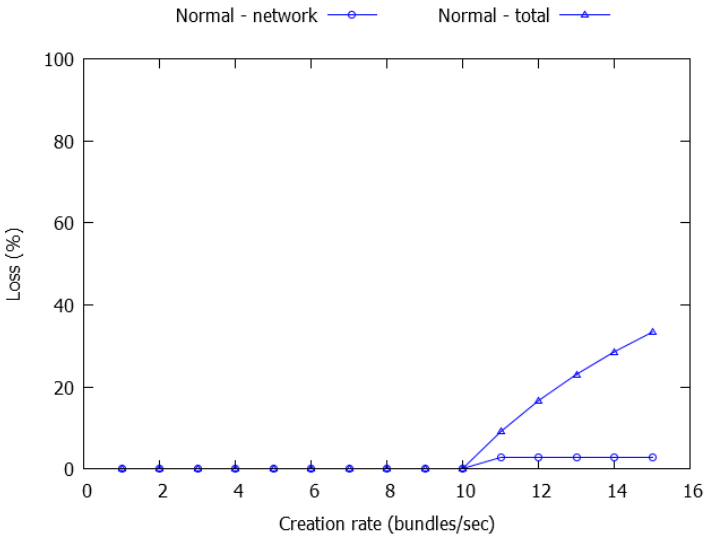
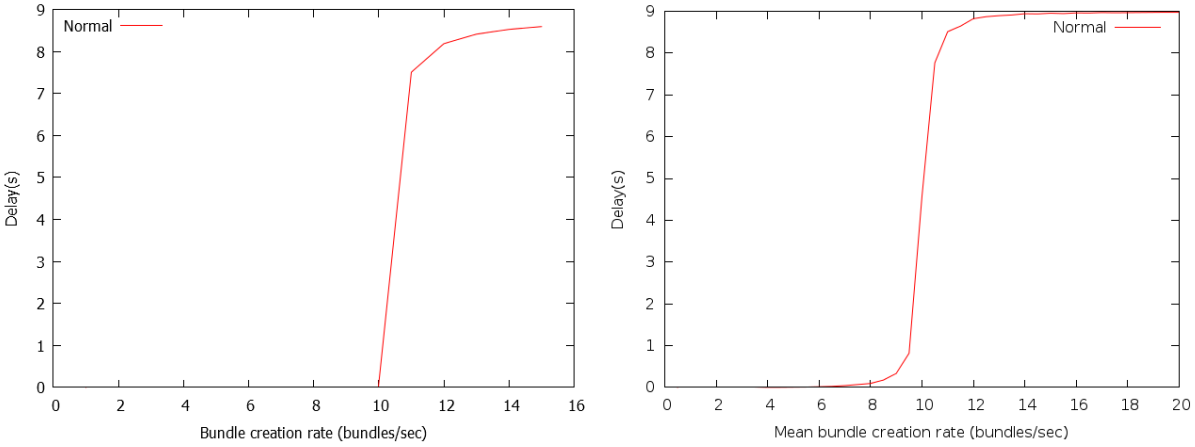


Figure 6.2: Loss of normal ORWAR without transmission priorities and constant bundle creation rate for the stationary mobility model after applying periodic KDM updates



(a) (b)

Figure 6.3: Mean bundle delay in normal ORWAR without transmission priorities with deterministic bundle creation rate (a) and Poisson distributed creation rate (b) for the stationary mobility model

When we plot the mean delay per bundle in normal ORWAR without transmission priorities using the static mobility model, we obtain Figure 6.3. Note that in this version we have already implemented the improvements regarding the propagation of the KDM-vector discussed previously.

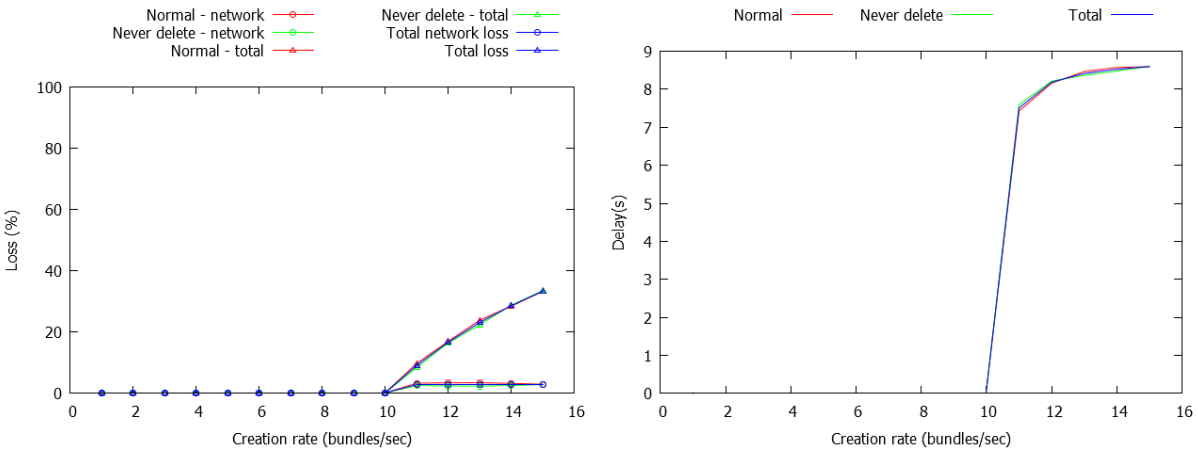
This figure can again easily be explained using the characteristics of the stationary model. As stated before, if $c \leq r_b$, then all created bundles can be transmitted as soon as they are created, resulting in zero delay. For values where $c > r_b$, the delay quickly increases as the buffer fills up and bundles will have to wait before being transmitted, as displayed in Figure 6.3a. This delay is however upper bounded by either the TTL of bundles or the maximum time a bundle has to stay in the buffer before being transmitted, depending on the buffer size and throughput of the medium.

In our example, we have a buffer size of 100 bundles and a throughput of 10 bundles/sec. Presuming $c > r_b$, once the buffer is full at time t, it will have 90 bundles remaining at $t + 1$. Any bundle injected at this time, will be transmitted after 9 seconds. As the model is stationary, no extra delay is introduced after transmission and hence the delay's upper bound is 9 seconds as this delay is smaller than the bundles' TTL. The delay in this network can therefore be given by:

$$D = \max\left(\frac{s_b - r_b}{r_b}, TTL\right) \tag{6.4}$$

with D denoting the mean delay on the network, s_b the buffer size in bundles and r_b the per-hop transmission rate in bundles/sec.

As we replace the deterministic constant bundle creation rate with a Poisson distributed creation rate, we observe that a delay is already introduced when $c < r_b$. This is easily explained by the fact that due to the use of a Poisson distribution, occasionally more bundles are injected than can be immediately transmitted, introducing delay. However, this should not happen often enough in quick succession to fill



(a) (b)

Figure 6.4: Loss (a) and mean delay (b) using the stationary mobility model and deterministic bundle creation for an equal distribution of Normal and Never Delete traffic

up the buffer, hence keeping the mean delay low.

Individual retention class behaviour in combination with Normal bundles

Now that we have established the behaviour of normal ORWAR without transmission priorities using the stationary mobility model, we can add traffic from the different retention classes and see how this affects the behaviour. First, we compare the Normal retention class to Never Delete. As we adjust the percentage of Never Delete bundles compared to Normal bundles, we see no change in behaviour compared to normal ORWAR, as evidenced in Figure 6.4

In this figure, we distribute the traffic equally between the Normal and Never Delete retention classes. This figure is representable for all possible distributions of Normal and Never Delete as the distribution does not have any affect on the behaviour.

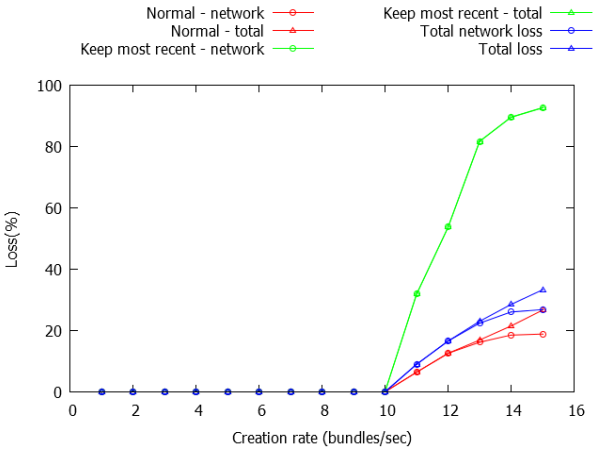
The explanation for this behaviour is easy. In Section 2.1.2 we have stated that Normal and Never Delete bundles can only delete bundles with a lower marginal utility. As transmission priorities are equal for every bundle, neither a Never Delete bundle or a Normal bundle is able to remove bundles from the buffer. Therefore they are equal in this setup and hence the behaviour will be similar to the case where there is only one retention class.

Differences to the original ORWAR without transmission priorities *can* be seen, however, once the Keep Most Recent retention class is introduced. Figure 6.5 shows the measured loss on the network for different distributions of Normal and Keep Most Recent traffic. Looking at Figure 6.5a, we notice there is a big difference in loss between the Normal and Keep Most Recent classes. At a creation rate of 14 bundles/second, the loss for Keep Most Recent traffic is already 90%, where the normal ORWAR scenario, depicted in Figure 6.2 shows a loss of just under 30%.

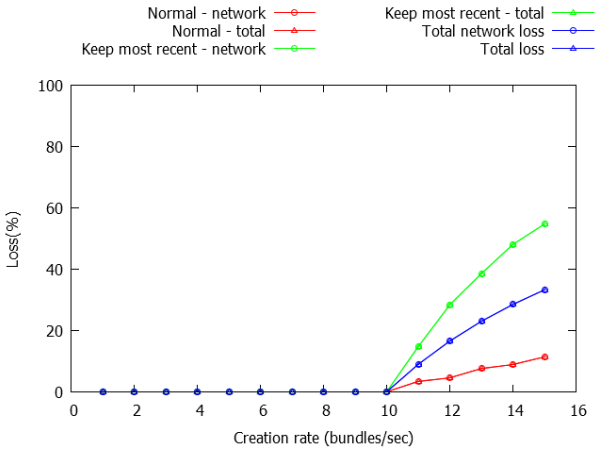
This difference can be explained through the use of stream IDs. In our setup, we use 16 distinct stream IDs, of which 4 can also be used by keep most recent classed traffic. In case any bundle is present in the buffer matching the stream ID of a newly created bundle, created at a later time than the existing bundle, the existing bundle is deleted. This means that with our stream ID distribution, 75% of the Normal bundles will not be deleted through this mechanism. As there is no other way to delete bundles upon insertion, these bundles will always arrive, similar to the original ORWAR scheme.

Likewise, for every Keep Most Recent bundle in the buffer, there is a 25% probability it will be deleted as soon as a new Keep Most Recent bundle is inserted. This means that as soon as the creation rate increases, the probability of being deleted from the buffer by a new bundle also increases. In addition, in the case where there is a low percentage of Keep Most Recent bundles compared to Normal bundles, this effect is even more prominent. There is a low probability that a Keep Most Recent bundle is able to delete a Normal bundle, resulting in these Normal bundles being normally injected into the source buffer. This results in the buffer filling up with bundles and therefore newly inserted Keep Most Recent bundles may not be transmitted immediately after insertion. Instead, they are forced to wait in the buffer until they are served. However, the longer a Keep Most Recent bundle has to wait in the buffer, the greater the probability it will be deleted by a newer bundle.

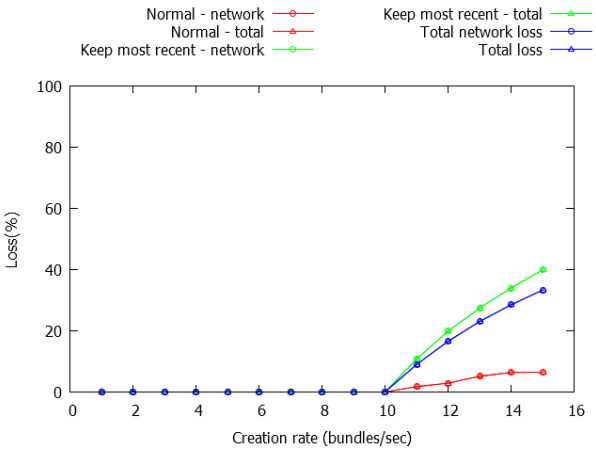
It is this difference in stream IDs that also explains why the Normal bundles are the only type that show a difference between total and network loss. The probability of a Keep Most Recent bundle not being inserted into the source buffer because it is full is fairly small as it will likely find an older bundle to delete and replace. Therefore the loss for this bundle type is mainly network loss. The Normal bundles however, cannot delete other bundles to insert themselves and may be dropped once the buffer is full.



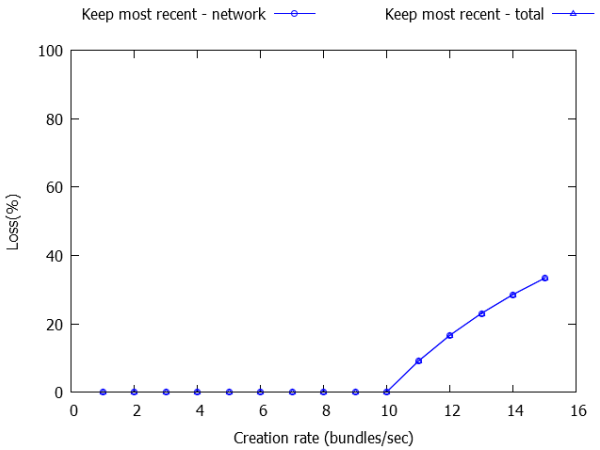
(a) Normal 90%, Keep Most Recent 10%



(b) Normal 50%, Keep Most Recent 50%



(c) Normal 20%, Keep Most Recent 80%



(d) Keep Most Recent 100%

Figure 6.5: Measured loss using the stationary mobility model and deterministic bundle creation for different distributions of Normal and Keep Most Recent traffic

This occurs later than in normal ORWAR however, as the Keep Most Recent bundles delay the buffer filling up by deleting obsolete bundles. Similarly, the network loss is higher for Normal bundles compared to Figure 6.2, as some Normal bundles may be deleted by a Keep Most Recent bundle matching its stream ID. When introducing Keep Most Recent bundles, 25% of the injected Normal bundles are at risk of being deleted by a more recent Keep Most Recent bundle whereas when only using Normal bundles, there is no way to delete bundles once they are injected if transmission priorities and bundle sizes are kept equal.

As we see the percentage of Keep Most Recent bundles increasing, we see the loss of these bundles decreasing and finally becoming similar to normal ORWAR, however without a distinction between total and network loss as explained above. When the percentage of Keep Most Recent bundles increases, the probability of transmitting Keep Most Recent bundles as soon as they are created increases. As there is an end-to-end connection, these bundles can not be deleted at intermediate nodes as they will not be stored at these nodes long enough. Only bundles remaining in the source buffer are at risk of being deleted by newer bundles. There is no distinction between network and total loss anymore as the buffer does not get the opportunity to fill up and new bundles can overwrite obsolete ones. This results in newly created bundles not being rejected by the buffer anymore.

The total loss on the network does not change in any of the tested distributions of Normal and Keep Most Recent traffic. This is due to the fact that the total loss on the network shows the maximum capacity of the network. This loss is caused by more bundles being created per second than can be transmitted, something no distribution of retention classes can change. Note that Keep Most Recent bundles can only delete bundles older than themselves. If bundles are inserted with the same creation time and stream ID, neither can delete the other. Their behaviour therefore is similar to the other classes in this case. This also ensures that every second, enough bundles are inserted to use the full network capacity if $c \geq r_b$.

Keep Most Recent traffic however shifts the moment of bundle deletion from the back of the buffer by dropping new bundles to replacing bundles already in the buffer by a newer version. Through this deleting of obsolete bundles, we observe that the total loss of Normal bundles decreases as the percentage of Keep Most Recent traffic increases. This is due to the fact that the majority of Normal bundles will not be deleted by Keep Most Recent bundles, but will also not be dropped at the source buffer due to insufficient storage space.

Figure 6.6 shows the network delay using different distributions of these retention classes. What we see here is that the delay greatly decreases as the percentage of Keep Most Recent bundles increases. In addition, the delay of arriving Keep Most Recent bundles is always smaller than that of Normal bundles. We already established that for the stationary mobility model, in normal ORWAR, the delay is caused by bundles waiting to be transmitted by the source node. With the introduction of Keep Most Recent bundles, the waiting period in the source buffer is reduced as obsolete bundles are removed rather than kept in the queue. Due to this deletion of obsolete bundles, no network resources are wasted on propagating these bundles.

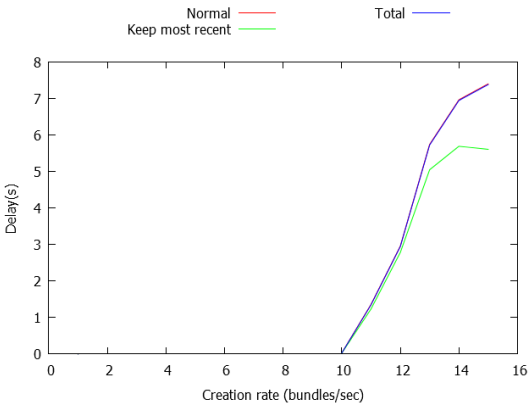
Furthermore, delay can only be measured for bundles that arrive at the destination. The longer a bundle waits in the buffer, the greater the probability it will be removed by a newer Keep Most Recent bundle. Therefore, only bundles which got transmitted relatively quickly after insertion will arrive at the destination and hence get their delay registered.

Retention class behaviour without Normal bundles

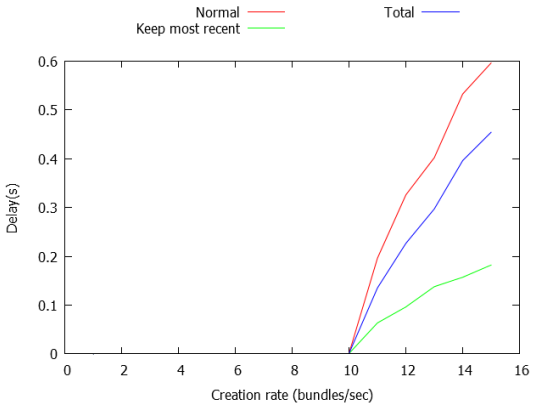
Lastly, we compare the Never Delete and Keep Most Recent classes with each other for different distributions. From the previous results we already concluded that Never Delete behaves similar to Normal in addition to Keep Most Recent being able to manage the buffer, reducing delay but increasing loss as it overwrites bundles. Figure 6.7 shows the loss and Figure 6.8 the delay on the network for the combination of Keep Most Recent and Never Delete classes.

In this figure we indeed see similar behaviour when comparing it to Figure 6.5. There are two differences worth noting, however. First, the split between network loss and total loss occurs at a lower creation rate when we compare figures 6.5a and 6.7a. Second, we see that the loss of Keep Most Recent bundles is even greater when coupled with Never Delete bundles compared to Normal bundles. In addition, the loss of Never Delete bundles becomes nearly non-existent as the percentage of Keep Most Recent bundles increases.

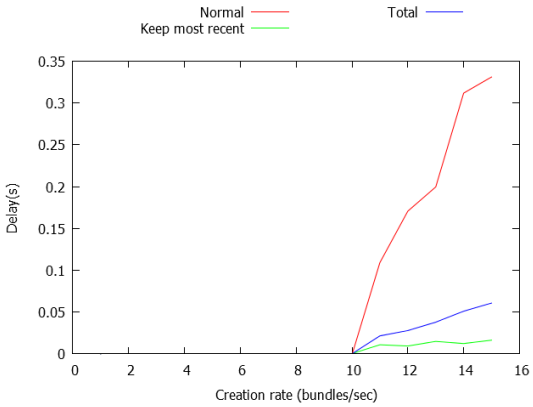
Every bundle has a stream ID, and Never Delete bundles are no exception. However, while Keep Most Recent bundles are allowed to delete obsolete Normal bundles with matching stream IDs, they are not allowed to delete Never Delete bundles. As a result, if a Keep Most Recent bundle removes part of the existing buffer by deleting obsolete bundles, any newly inserted Never Delete bundles will take up that space until its TTL expires or it is delivered at the destination and an acknowledgement is received. In the end, with either a high enough percentage of Never Delete bundles or high enough creation rate, this means that the buffer will slowly become full with Never Delete bundles. Once the buffer is full with this type of bundle, any newly created bundle will be dropped, resulting in the split between network and



(a)
Normal 90%,
Keep Most Recent 10%

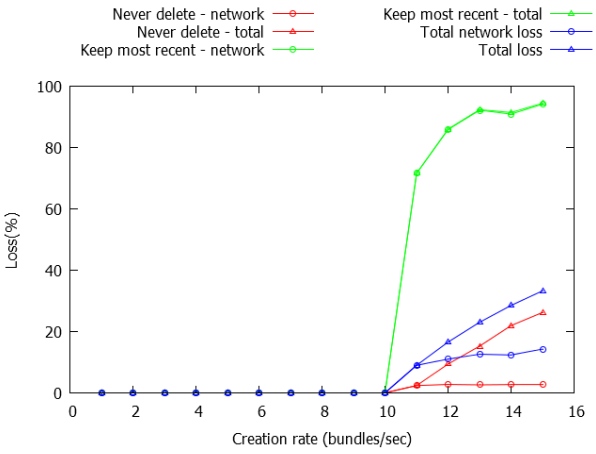


(b)
Normal 50%,
Keep Most Recent 50%

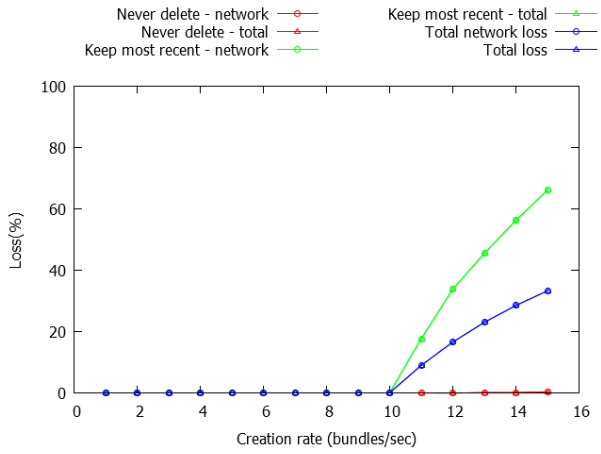


(c)
Normal 10%,
Keep Most Recent 90%

Figure 6.6: Measured delay using the stationary mobility model and deterministic bundle creation for different distributions of Normal and Keep Most Recent traffic



(a) Never Delete 90%, Keep Most Recent 10%



(b) Never Delete 50%, Keep Most Recent 50%

Figure 6.7: Measured loss using the stationary mobility model and deterministic bundle creation for different distributions of Never Delete and Keep Most Recent traffic

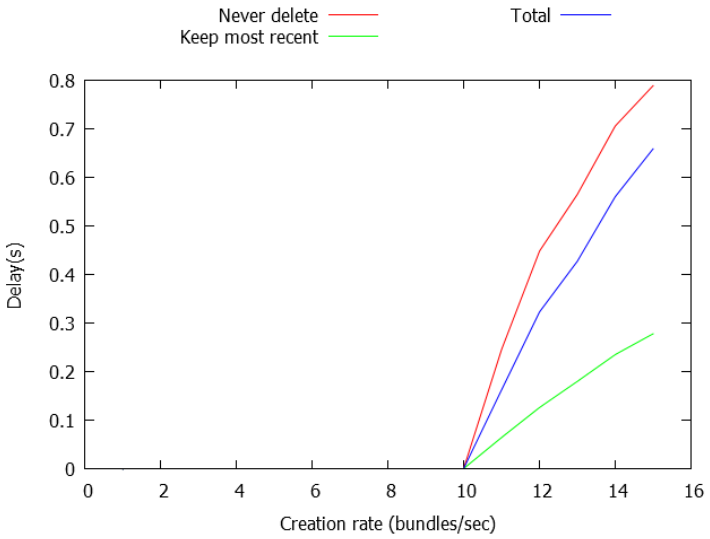


Figure 6.8: Mean delay using the stationary mobility model and deterministic bundle creation for an equal distribution of Never Delete and Keep Most Recent traffic

total loss.

This phenomenon also explains why the loss of Never Delete bundles is almost non-existent as the percentage of keep most recent bundles increases. As the Keep Most Recent bundles keep overwriting each other, freeing space in the buffer before it is full, the Never Delete bundles will always be inserted. As once inserted they cannot be removed before they arrive at their destination or they expire, their loss will be close to 0%, with the only loss being the buffer remainder after the simulation is completed.

In Figure 6.8 we see similar behaviour to Figure 6.6b. The delay is however slightly higher in this scenario than in the scenario using Normal and Keep Most Recent bundles. Again, this is explained by the fact Keep Most Recent bundles are not allowed to delete Never Delete bundles. The buffer will therefore not be cleared as much as in the case where Normal bundles are used instead of Never Delete bundles. This increases the delay as bundles have to wait in the buffer for a longer time before being transmitted.

6.1.2 Linear mobility model

Introducing mobility to our simulations will show the influence of intermittent connectivity on our retention classes. In this scenario, traditional transport protocols will cease to function effectively as there often will not be an end-to-end connection between nodes A and C. Our goals are still similar to the stationary model however, in that we aim to analyse the basic behaviour of ORWAR and the influence of our retention classes on this behaviour.

Basic ORWAR performance

As we change the mobility model from the stationary model to the linear mobility model, we observe some important changes in behaviour. First, there is a change in the observed loss compared to the stationary model, as shown in Figure 6.9. In this figure we notice that a total loss $> 0\%$ occurs at an earlier moment than in the stationary model. Rather than loss occurring once the creation rate exceeds the transmission rate, bundles are dropped before network injection at an earlier time. We know they are dropped before injection as the total loss increases while the network loss does not.

This behaviour is caused by the movement of node B and the effect this has on the propagation of acknowledgements and subsequent bundle removal from the buffer. This model also illustrates the need for a retention time for acknowledgements as discussed in Section 2.3. The retention time in effect serves the purpose of adding a delay between a time-out and retransmission of a bundle. What happens here is that it takes a longer time before node A can receive an acknowledgement for its transmitted bundles through the KDM-vector. This in turn results in bundles staying in the source buffer for a longer time, filling up the buffer.

We can easily explain this through an example scenario. In our model, node B stays in range of either node A or C for eight seconds. Presume a bundle b_X is inserted at node A just as node B moves out of range. In addition, we assume that at this point of time, the buffer at node A is completely empty. As node B moves out of range, b_X will be placed at the front of the buffer. It now takes eight seconds for B to move back into A's transmission range. At this point, b_X will be transmitted to B. However, in contrast to the stationary model, there is now a delay before the bundle is transmitted to C, as B needs

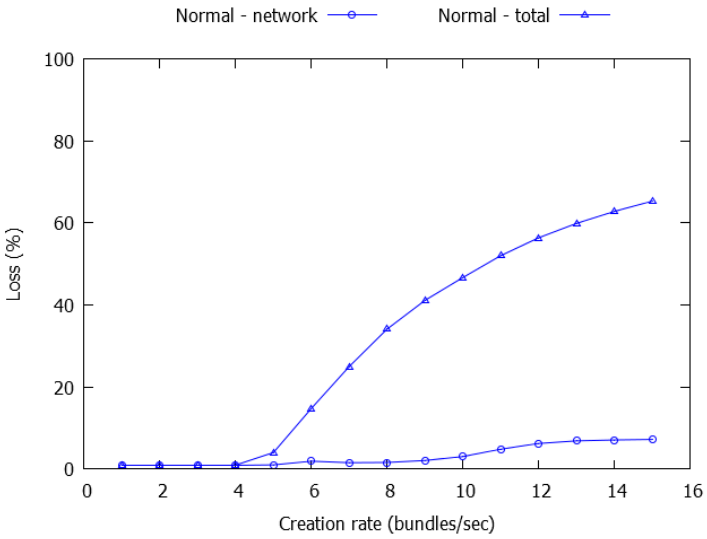


Figure 6.9: Loss in normal ORWAR without transmission priorities and constant bundle creation rate for the linear mobility model

to move back into C's transmission range. Therefore, it takes another eight seconds before b_X is finally delivered to node C.

Now here is where the "unnecessary" occupying of node A's buffer by b_X occurs. Before node A can receive the acknowledgement that b_X has been successfully delivered and therefore clear it from its buffer, it has to wait yet another eight seconds before node B moves within range again. Adding all these delays together, this means that it takes 24 seconds between b_X's creation and a received acknowledgement for this bundle. However, its TTL is only 20 seconds and it will therefore be deleted before an acknowledgement can arrive.

We can see from this situation that even with a low creation rate, bundles can remain in the buffer for the complete duration of their TTL, depending on the interval at which the source node is serviced by other nodes, regardless of whether these bundles are actually delivered to the destination. As a result, the source buffer can fill up at a lower load than in the stationary model, where the only factor is the network's transmission rate. Consequentially, this means that the source buffer may need to drop incoming bundles sooner than in the stationary model. The storage space reserved for bundles should therefore be calibrated based on the network connectivity, including the rate at which nodes are serviced by others.

This situation also displays the need for the aforementioned retention time τ. In this example we notice that while the bundle is successfully delivered, the acknowledgement will arrive too late. It takes node B 24 seconds in total for a round-trip to deliver the acknowledgement of b_X to node A, while its TTL is only 20 seconds. Without the retention time, this acknowledgement will be deleted from the KDM-vector before node B reaches node A again, as in this case acknowledgements are only stored for the duration of the TTL of the corresponding bundle. Without the retention time, node A might attempt to retransmit a successfully delivered bundle, wasting resources. Of course this situation only arises if DTN is configured to retransmit bundles it has not received an acknowledgement for.

Note that just adding this retention time to the KDM-vector is insufficient. In the bundle protocol specification [5], the bundle ID is formed by the source EID and creation timestamp rather than a separate field. A retransmission will therefore have a different ID compared to the original bundle. This retention time should therefore be integrated at the source node as well, though the bundle itself may be deleted already in accordance to its TTL. This way, the source node should be able to receive an acknowledgement for a successfully delivered bundle, without increasing the TTL of bundles, to prevent unnecessary retransmissions. In other words, not only bundle transmission, but also acknowledgement-propagation should be delay tolerant. To simplify our simulations, we do not simulate retransmissions, however this situation does show a point of concern which can be detrimental to performance.

This situation therefore leads us to some conclusions based on the network density/mobility. The sparser connected a network is in both density and mobility, the larger the storage buffer of a node should be. In addition, the combination of TTL and retention time of a bundle should be larger in these networks. These parameters should be chosen in such a way as to accommodate the round trip of a bundle.

The mean delay in the linear model is also different from the stationary model, as evidenced in Figure 6.10. In the stationary model, the delay is determined by the buffersize, creation rate, transmission rate and TTL of bundles. In summary, as soon as the creation rate exceeds the transmission rate, the delay is upper bound by either the time it takes for a bundle to reach the front of the buffer or the TTL of the bundle, whichever is lower. This upper bound is reached shortly after the creation rate exceeds the

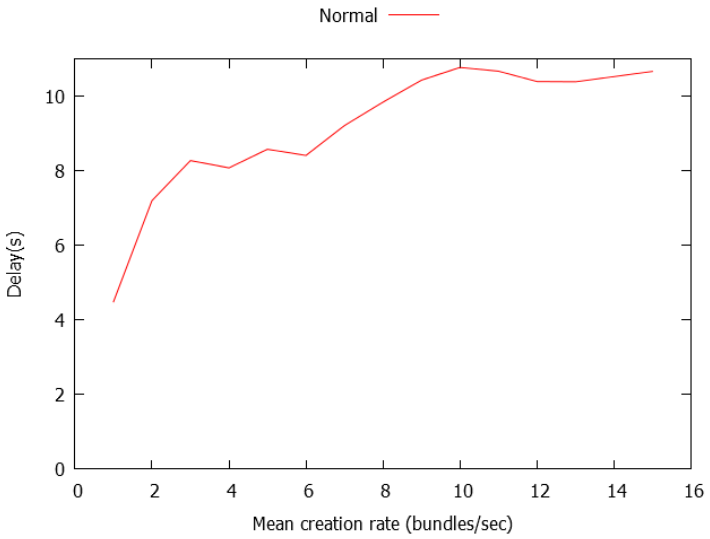


Figure 6.10: Delay in normal ORWAR without transmission priorities and constant bundle creation rate for the linear mobility model

transmission rate. In the linear and non-linear mobility models, the movement of node B plays a central part in the delay.

When mobility for node B is introduced, bundles cannot be directly transmitted from source to destination anymore, except when node B is directly between nodes A and C. In all other situations, bundles transmitted from node A to node B need to wait before node B gets within transmission range of node C before they can be transmitted to their destination. Likewise, bundles injected at node A need to wait before node B moves within range before they can be transmitted. This introduces extra delay as node A waits for node B to move within range.

However, the introduction of mobility also means that node B is either just a receiver or just a transmitter, depending on which node it is associated with. This is in contrast to the stationary model where at each timeslot node B performed both roles: receiving from node A and transmitting to node C in the same timeslot. This results in the full network transmission capacity of 20 bundles/sec being reached for a single hop if node B is associated with only one other node.

Whether the source buffer completely fills up is therefore dependent on a couple of factors. The first factor is the buffersize of both nodes A and B. Whereas in the stationary model the buffersize of node B is not important as bundles get immediately forwarded, now it has to be big enough to accommodate all the bundles which node A transmits before it moves on to node C. In addition, node A needs to be able to accommodate the influx of bundles while node B is out of range. The advantage of this model however, is that as soon as node B is within range of node A again, it will receive bundles from node A's buffer at the channel's maximum transmission rate, in our case 20 bundles/sec. Node A however has to wait another complete round-trip of node B before it receives acknowledgements for these bundles, clearing them from the buffer, as explained earlier.

Another factor of course is the TTL of a bundle. If the bundle's $TTL * c <$ buffersize, with c denoting the bundle creation rate, node A's buffer will never fill up as bundles will time-out before the maximum capacity is reached.

The final factor is how long node B stays out of range of node A and within range of node C. In our model, we have chosen equal durations for both of them for simplicity. The longer node B stays away from node A, the more bundles get created at node A with nowhere to go. This will result in either the buffer filling up and new bundles being dropped or a situation where bundles will reach their TTL and get deleted if the buffersize is adequately large. In addition, node B needs enough time within node C's range so that it can transmit all the bundles it has received from node A. Should this time in node C's range be too little, it will require several round-trips to deliver all the bundles from node A. This in turn will greatly increase the delay and may also affect the loss as its buffer will not be empty upon its return to node A. Bundles may also expire more often in this situation due to the longer delay.

Due to these factors, the delay shows a more gradual increase over time before reaching its upper bound. This upper bound is determined by the average waiting time of a bundle residing at node A before node B moves within range and the rate at which node B can transmit bundles to node C. Specifically, whether node B can transmit its complete buffer to node C in a single trip.

What we also see is that the delay does not monotonously increase but has two more or less stable levels, one level with a delay of roughly 8 seconds for $3 \leq c \leq 6$, and one level with a delay of roughly 10.5 seconds for $c \geq 10$. These levels are caused by the mobility of node B. The delay is always measured by the receiving node by comparing the creation timestamp with the delivery timestamp. This means that even though bundles may expire at the source, their delivery delay is still measured.

When node B moves between nodes A and C, there is a small timeframe where it is in range of both nodes and an end-to-end connection exists. This also means that at this point the network behaves as in the stationary model, with a transmission rate of 10 bundles/sec from node A to node C. If less than 10 bundles have been created while node B was out of node A's transmission range, all nodes can immediately be transmitted when it moves back into range. This occurs at $c = 1$, resulting in 8 bundles being created and therefore an average delay of 4.5 seconds.

As c gets larger, delay is added as bundles need to wait for node B to move within range of C again. As bundles are served in a FIFO manner, the average waiting time will be roughly equal to the time it takes for node B to move within range of either node A or node C again. In other words, the delay will be around 8 seconds. Minor fluctuations are possible as node B's transmission speed is increased when it is in range of only one other node, meaning it might take a second more or less to empty its buffer contents depending on the exact number of bundles contained when entering node C's range.

Delay starts to increase even further if c is so high that within a single round-trip of node B, loss starts to occur due to bundles not being inserted. However, node B is still able to transmit its complete buffer contents when it is within range of node C. In this scenario, the average delay will increase as bundles which would normally have a lower delay due to being inserted later will not be inserted at all as the buffer will be full. The bundles which are actually inserted on the network and subsequently delivered are therefore the ones that have to wait the longest for their delivery, increasing the average delay. Hence why we see the delay increasing once $c > {}^{s_b}/_{RTT_B}$, with s_b the buffer size in bundles and RTT_B the round-trip time of node B. In our case this evaluates to $c > {}^{100}/_{16} = 6.25$.

Retention class influence

Similar to the stationary model, we see no change in behaviour between Normal and Never Delete bundles. Again, as transmission priority is identical and neither is allowed to delete the other from the buffer, behaviour is similar.

We do see some change between Keep Most Recent and Normal bundles, however, compared to the stationary model. As shown in Figure 6.11, we see some similarities to the stationary model when the Normal bundles outweigh the Keep Most Recent bundles. The Keep Most Recent bundles still show a larger loss and the Normal bundles still show a split between network and total loss. However, as we increase the percentage of Keep Most Recent bundles, the behaviour changes compared to the stationary model. Due to the delay of bundles being transmitted between nodes A and C, the probability of an obsolete bundle being overwritten by a newer one increases rapidly. This will free up space for Normal bundles, which will rarely get dropped due to a full buffer. The only loss that occurs for Normal bundles is the loss introduced by newer Keep Most Recent bundles matching its stream ID. Therefore, the loss for both these classes will reach a stable level at even a low creation rate. This rate is near 95% for Keep Most Recent bundles and 20% for Normal bundles. These metrics depend on the mobility of node B. When node B makes more frequent, shorter duration round trips, the loss should be lower. Bundles will be stored for a shorter time and therefore are at a lower risk of being deleted by a newer Keep Most Recent bundle.

What is also noticeable, is that even the loss for Normal bundles is already more than 0% even at a low creation rate. Again, this is due to bundles not being propagated fast enough due to node B's mobility. This means that even at a low creation rate, channel resources may be used for bundles which will get deleted at a later point. One of the reasons to use ORWAR is its efficient use of the limited medium resources. Therefore, the proportion of Keep Most Recent bundles in regard to the total traffic load should be carefully moderated to prevent this behaviour from occurring too much. Especially at a low creation rate and when there are longer periods between topology changes, the number of Keep Most Recent bundles should be limited.

In terms of delay, we observe that as long as the percentage of Keep Most Recent bundles $50\% \leq \%KMR < 100\%$, the delay of the Keep Most Recent bundles stays relatively stable for all values of the bundle creation rate c, while the delay of Normal bundles increases as c increases. Figure 6.12 shows this delay for different distributions of Keep Most Recent and Normal bundles. This is caused by Keep Most Recent bundles constantly deleting older Keep Most Recent bundles due to the delay introduced by node B's mobility. When bundles are constantly deleted by newer copies, only the most recent injected bundles will reach the final destination, resulting in a stable delay regardless of the creation rate. The Normal bundles will still increase in delay as the majority of these will still not be deleted by Keep Most Recent bundles, therefore they will have to wait in the buffer before the older bundles are transmitted. It is also for this reason why there is still a delay for Keep Most Recent bundles, as they will have to wait for these earlier inserted Normal bundles to be transmitted as well. When all bundles are classified as Keep Most Recent bundles, we see a rapid decline in delay as the creation rate increases, as at a high enough c, every timeslot the complete buffer will be wiped and replaced by the newly created bundles, creating a bound on delay. The only bundles that will arrive in this scenario are the ones generated when node B is right between nodes A and C, resulting in an end-to-end connection. We also see this bound differs very little from the stationary mobility model.

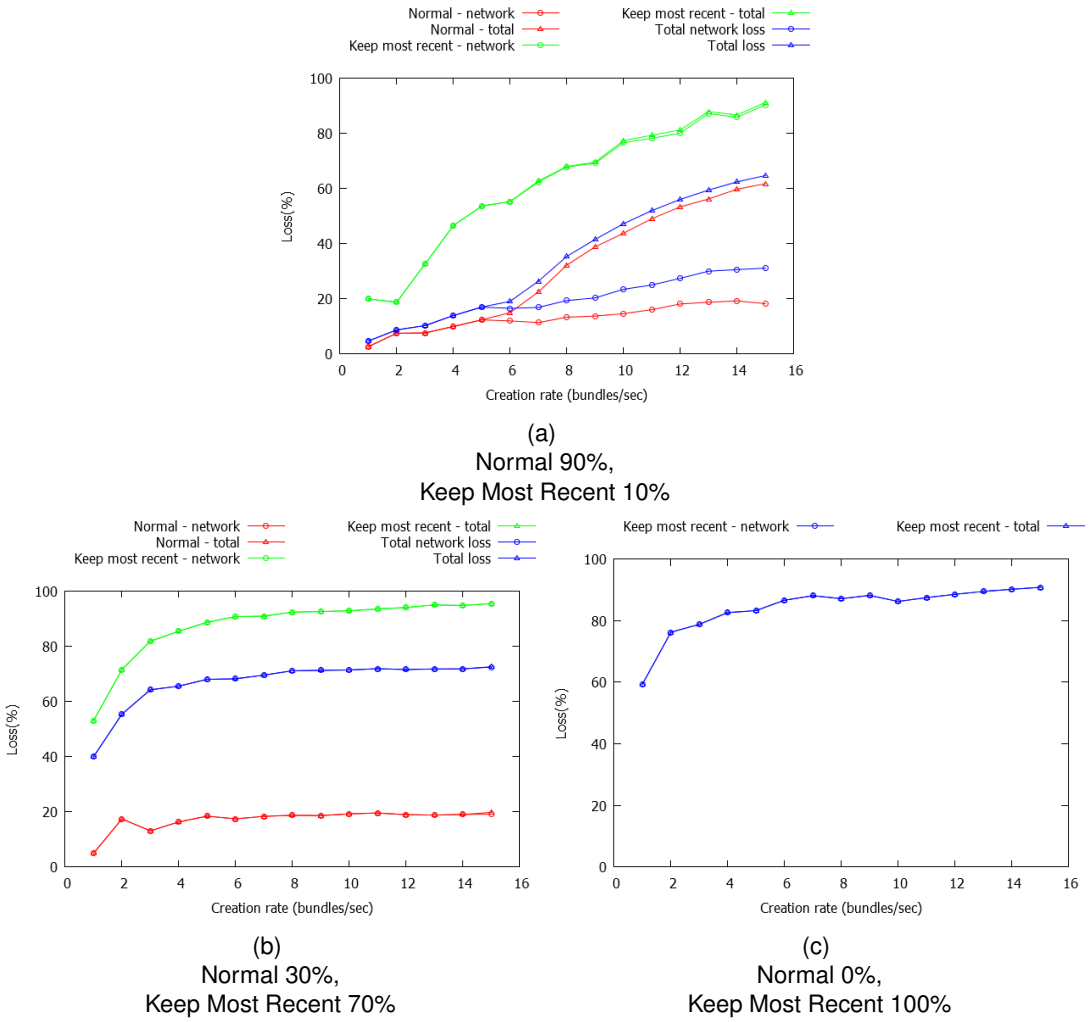
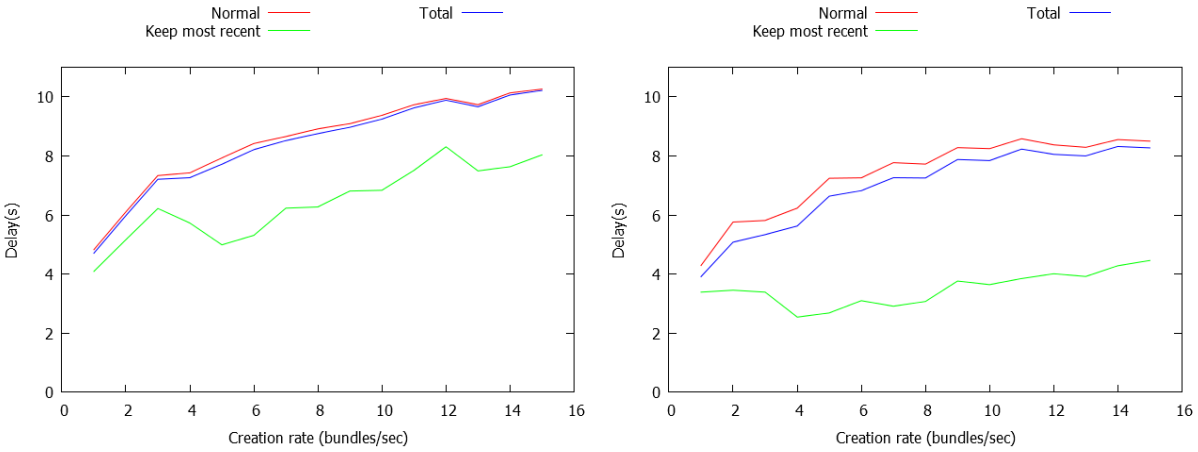


Figure 6.11: Measured loss using the linear mobility model and deterministic bundle creation for different distributions of Normal and Keep Most Recent traffic



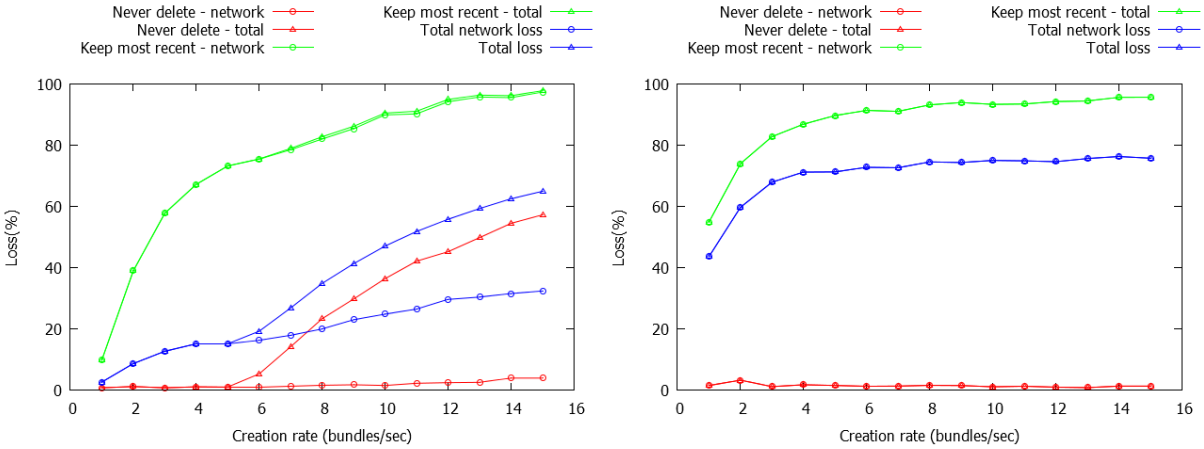
(a) Normal 80%, Keep Most Recent 20%

(b) Normal 50%, Keep Most Recent 50%

(c) Normal 30%, Keep Most Recent 70%

(d) Normal 0%, Keep Most Recent 100%

Figure 6.12: Measured delay using the linear mobility model and deterministic bundle creation for different distributions of Normal and Keep Most Recent traffic



(a) Never Delete 80%, Keep Most Recent 20%

(b) Never Delete 20%, Keep Most Recent 80%

Figure 6.13: Measured loss using the linear mobility model and deterministic bundle creation for different distributions of Never Delete and Keep Most Recent traffic

Comparing the Keep Most Recent and Never Delete bundles, we again see similar behaviour as comparing the Normal and Keep Most Recent bundles, shown in Figure 6.13. Similar to the stationary model, the Never Delete bundles show no network loss whereas the Normal bundles do show network loss. Different from the stationary model, however, is that in this case, the loss of the Keep Most Recent bundles does not increase when comparing similar distributions of Keep Most Recent and Never Delete bundles to Keep Most Recent and Normal bundles. In this case, less medium resources are wasted for unnecessary transmissions compared to the distribution with only Normal and Keep Most Recent bundles. The delay shows similar behaviour to the distributions of Keep Most Recent and Normal bundles.

6.1.3 Non-linear mobility model

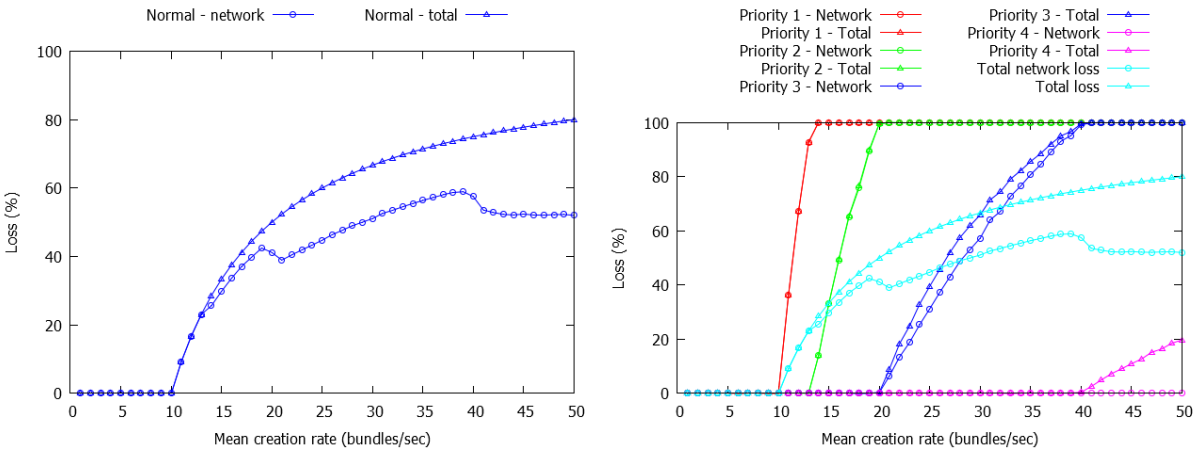
The last model we analyse is the non-linear mobility model. The ORWAR protocol reverts to best-effort transmissions once it is in contact with a node for which its calculated contact window has already expired. In the case of best-effort transmission, ORWAR ceases to try to reorder its transmission buffer based on the utility-per-bit and will start transmitting all bundles in the buffer in sequence. When we only use retention priorities and an uniform size, bundles are transmitted in a FIFO manner as the utility-per-bit is equal for all bundles. This also means that best-effort transmission is similar to ORWAR's ordered transmission using only retention classes. The contact windows between the nodes is kept equal in both the linear and non-linear mobility models. As a result, the behaviour in the non-linear model is equal to the linear model when we use bundles of equal size and transmission priority.

6.2 Retention classes with transmission priorities

6.2.1 Stationary model

Now that we have a basic comparison of the influences of the different retention classes on network metrics, it is time to observe what happens when transmission priorities are added. Bundlesizes are still kept equal, therefore a bundle's transmission priority is the only factor influencing its place in the transmission buffer; high priority bundles will always be transmitted before low priority ones. The addition of these priorities however, allows bundles to delete bundles of a lower priority in case the buffer fills up. In addition, unlike the previous scenario, bundles are not transmitted in an absolute FIFO manner anymore. Within each transmission priority however, FIFO scheduling still occurs as bundle sizes are kept equal and thus the marginal utility within a priority is equal for all bundles.

The behaviour shown when adding transmission priorities will already be close to normal ORWAR, though there is a limited number of marginal utilities, resulting in little variation in bundle scheduling as the lowest priority bundles will always be transmitted last. In turn, this may result in starvation of low priority bundles. The bundlesize is kept equal however for simplicity. In these experiments we analyse what influence the combination of priority scheduling and retention classes has on network performance. While we used a deterministic approach in the previous section to determine behaviour, we now use Poisson distributed creation rates for more realistic results.



(a) Loss per retention class

(b) Loss per transmission priority

Figure 6.14: Loss in normal ORWAR with transmission priorities and constant bundle size for the stationary mobility model

Basic ORWAR performance using constant bundle sizes

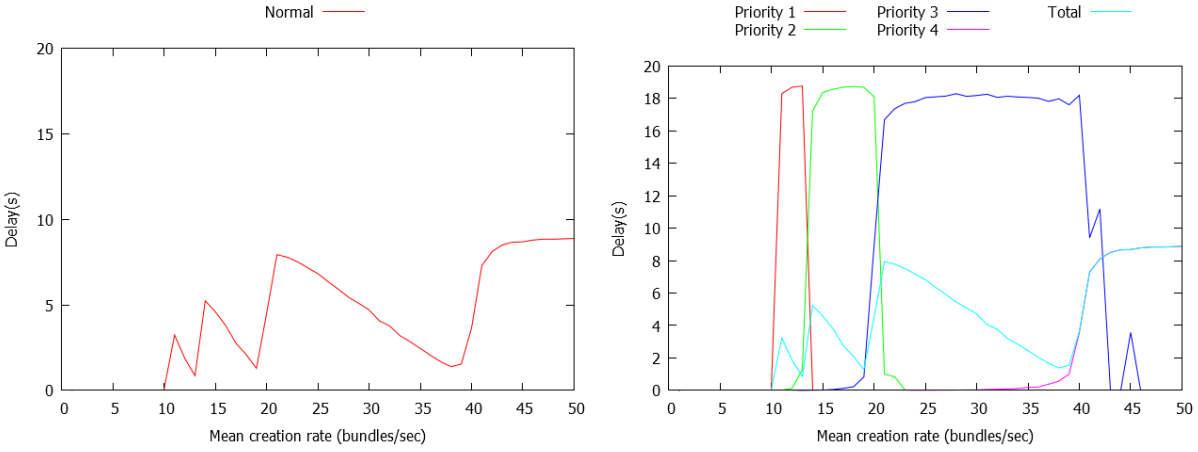
In Figure 6.14 the loss in normal ORWAR with transmission priorities in the stationary mobility model is shown. When we compare Figures 6.2 and 6.14a we notice some differences introduced by the transmission priorities. While loss only starts to occur at a creation rate ≥ 10 similar to the case without transmission priorities, we also notice that the split between total loss and network loss occurs at a later time in Figure 6.14a. At first, the total loss is equal to the network loss as low priority bundles get replaced by higher priority bundles. This ensures there always being enough space in the buffer for new bundles to be injected. New bundles will therefore not get dropped before insertion, hence the total loss equalling the network loss. This behaviour is also evidenced in Figure 6.14b where it is shown that the first loss that occurs is only due to lowest priority bundles being overwritten.

This behaviour continues until the loss of Priority 1 bundles reaches 100%. At this point, the total loss starts exceeding the network loss as these bundles will be dropped before injection once the buffer is full. We also start seeing loss of Priority 2 bundles at this point. For creation rates ≥ 13, Priority 1 bundles will be inserted at first, but never transmitted due to the limited bandwidth. Higher priority bundles will be inserted in front of these bundles, effectively starving the already existing low priority bundles. After a while, the tail of the buffer will be full of Priority 1 bundles. Newly generated Priority 1 bundles will be dropped and existing bundles will be replaced by higher priority bundles as they are generated. Therefore, the difference between total loss and network loss is based on the number of Priority 1 bundles dropped.

We see this behaviour repeat itself at a creation rate ≥ 20, when in addition to Priority 1, also Priority 2 bundles are not delivered anymore and loss occurs of Priority 3 bundles. What is noticeable here however is the slight drop in network loss while the total loss increases. At a creation rate of 20 bundles/sec, on average there will be 5 bundles generated per second of each priority. We have already established that the effective transmission rate at node A equals 10 bundles/sec. This means that after a while, the first 10 bundles in the buffer will be of transmission Priorities 3 and 4, with the last 90 being of transmission Priority 2. Once this stage is reached, the order in which bundles are inserted determine whether bundles are dropped before injection or replaced after injection. Low priority bundles may be inserted, but will be immediately replaced by higher priority bundles. In addition, Priority 1 and 2 bundles are also at a high risk of being dropped as they are not allowed to replace existing bundles. The network loss metric will therefore be slightly lower as less bundles will be actually injected, skewing this value. A similar drop will occur as the creation rate ≥ 40, when also no Priority 3 bundles will be transmitted anymore and loss of Priority 4 bundles will occur.

The delay in this scenario is shown in Figure 6.15. Comparing this to the version without transmission priorities shown in Figure 6.3, we notice that the mean delay is significantly lower when transmission priorities are introduced. As evidenced in Figure 6.15b, however, this reduced mean delay comes at the cost of a very high delay for low transmission priorities. The dips in the mean delay coincide with the points where the total loss of a transmission priority nears 100%. At this point, the delay of these priorities cannot be measured and therefore the measured delay is only based on the higher priority bundles which will in general be transmitted with a lower delay.

Once the creation rate gets high enough however, the network will only be able to transmit Priority 4 bundles. As bundlesizes are kept equal, this means that the situation reverts to that of equal transmission priority and equal bundle size. In other words, the delay will reach its maximum value of 9 seconds as explained in Section 6.1.1.



(a) (b)

Figure 6.15: Mean delay in normal ORWAR with transmission priorities and constant bundle size for the stationary mobility model

What can also be seen in this situation, is that the higher the transmission priority, the longer the delay stays at its upper bound defined by the bundle's TTL. As long as the loss is not 100%, the delay can be measured. Higher priority bundles will reach the 100% loss rate at a slower pace, as bundles are deleted back-to-front. This means newly inserted low-priority bundles get deleted first if space has to be freed in the buffer. The higher priority a bundle has, the less bundles are allowed to delete it from the buffer as equal priorities cannot remove one another from the buffer. High priority bundles may therefore be transmitted with a high delay. However, as long as there are not enough higher priority bundles to delete or starve them from the buffer, they will be transmitted eventually. Once the creation rate is high enough such that only Priority 4 bundles are inserted, the mean delay will again be upper bound by 9 seconds, similar to Figure 6.3, as now all inserted bundles will have equal transmission priorities again.

Never Delete bundles in combination with Normal bundles

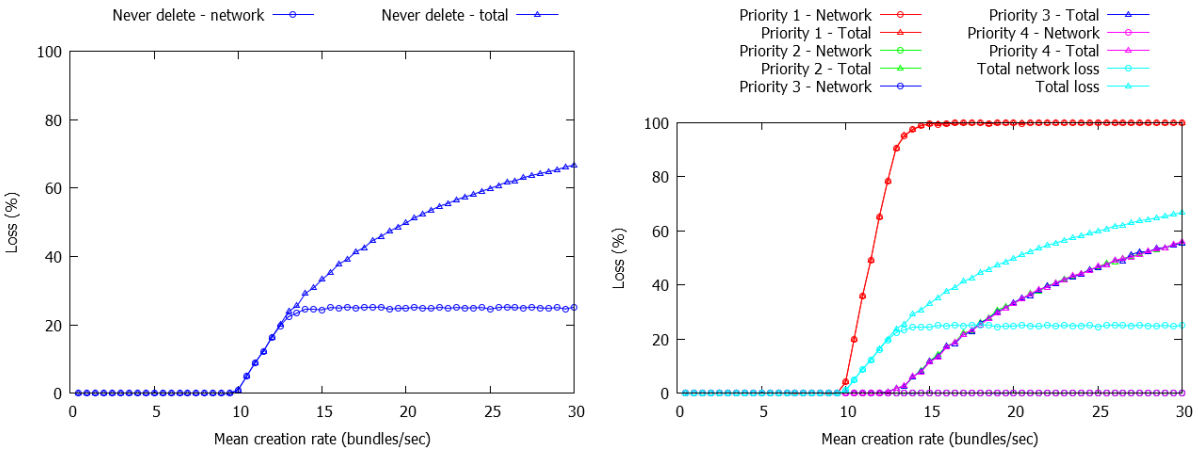
In contrast to the previous cases where we do not take transmission priorities in account, the Never Delete retention class shows different behaviour to the Normal retention class when transmission priorities are involved. The loss and delay statistics for a network only transmitting Never Delete bundles are shown in Figure 6.16. The total loss is again similar to the Normal retention classes. This is not surprising as the total loss is determined by the network capacity, in terms of transmission capacity, buffer capacity and the total number of bundles generated.

The network loss however is lower and does not increase as the load increases when we transmit only Never Delete bundles. Never Delete bundles show similar behaviour to the case where we only tested retention classes. In both cases, regardless of transmission priority, these bundles are not allowed to remove one another from the buffer. This means that as long as a bundle is inserted on the network, it will never be deleted by another bundle. This means that even high priority bundles may get dropped before insertion if the buffer is full.

The network loss, while at a stable level, is still higher than the network loss measured in Figures 6.2 and 6.4. In these previous measurements, the network loss was only caused by the bundles remaining in the buffer after our simulations have completed, meaning it would be 0% as $t \to \infty$. In this case however, there is some actual network loss occurring during the simulation.

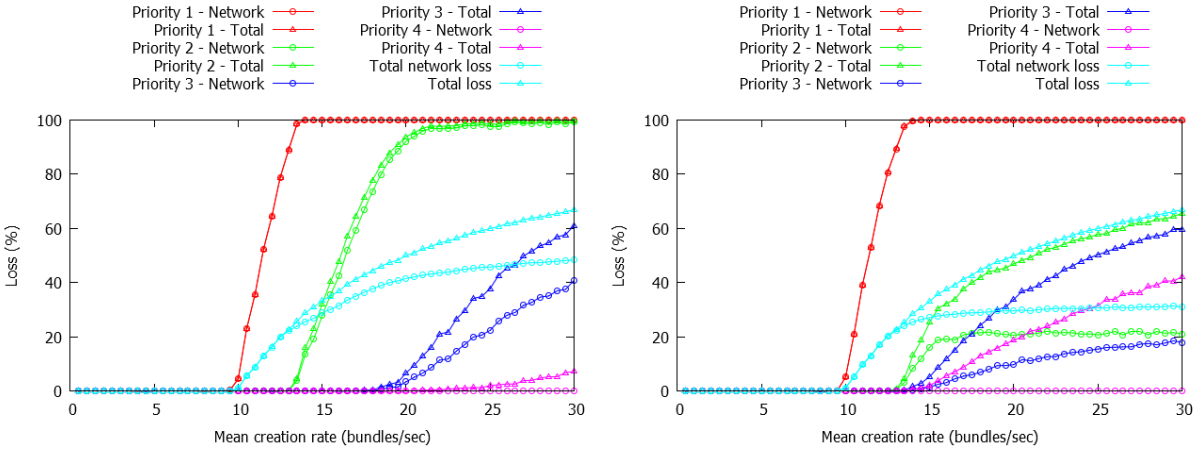
While never delete bundles may not be allowed to delete one another upon insertion, the transmission priorities still lead to bundles being ordered within the buffer. Bundles with a low priority can still be inserted presuming there is enough space in the buffer, however they will never be transmitted when the creation rate is high. As a result of this, these bundles will suffer from starvation and get deleted anyway as their TTL expires. This is evidenced in Figure 6.16b, where we observe the lowest priority bundles suffering from 100% loss.

In contrast to the Normal retention class, we see that Priority 2 bundles do not suffer a similar network loss increase as Priority 1 reaches 100% loss. Priority 1 bundles will always get inserted if there is free space in the buffer, but they will always subsequently starve. However, the time that they are still in the buffer, they will occupy storage space and cannot be removed by higher priority bundles. As the buffer fills up, new bundles need to wait for these bundles to starve before they can be inserted, regardless of transmission priority. This limits the insertion rate of new bundles once the buffer is full. As a result, there will never be enough high priority bundles inserted for Priority 2 bundles to suffer from similar starvation. In addition, as bundles of the same priority are served in a FIFO manner, any new Priority 2 bundles inserted reduce the waiting time of the oldest priority 2 bundles by reducing the



(a) (b)

Figure 6.16: Loss in ORWAR with transmission priorities and constant bundle size using only Never Delete retention for the stationary mobility model



(a) Never Delete 50%, Normal 50% (b) Never Delete 80%, Normal 20%

Figure 6.17: Loss per transmission priority in ORWAR with constant bundle size, using the stationary mobility model for different combinations of Never Delete and Normal bundles

number of available slots in the buffer for higher priority bundles. Should the buffersize increase and/or the transmission rate decrease, we will see similar starvation for other priorities.

In Figure 6.17 we see that increasing the percentage of Never Delete traffic decreases the loss of lower transmission priority bundles as the bundle creation rate c increases, with the exception of Priority 1 bundles. Especially the network loss is greatly reduced. However, this behaviour comes at the cost of increased loss for higher priority bundles and most loss being caused by bundles not being inserted on the network. As soon as Priority 1 bundles reach 100% loss, all other priorities start to show loss as they will not be inserted in the buffer. This is detrimental for the performance of high-priority bundles as they will now suffer from loss at a significantly lower creation rate.

The mean delay using Never Delete bundles is displayed in Figure 6.18. There is a significant difference whether or not Normal bundles are inserted as well. When 100% of the bundles is Never Delete, we notice a very low mean delay compared to normal ORWAR. In addition, there are always some Priority 1 bundles still received by the destination, as evidenced by the delay being unequal to 0. The network loss for these bundles is still very high however and this delay is likely measured from just a few bundles. As soon as we introduce Normal bundles as well, we see that these lowest priority bundles will suffer from 100% loss at a high enough c as some of these low priority bundles may now be overwritten by higher priority bundles.

When comparing Figures 6.18c and 6.18d, we notice that increasing the percentage of Never Delete bundles created reduces the mean delay in addition to reducing the delay of Priority 2 and Priority 3 bundles. This is caused by these bundles not being starved or overwritten and always being served eventually. The cost for this reduced delay however is the increased loss for these priorities caused by bundles dropped before they are inserted in the source buffer. Therefore we can conclude that *if* bundles are inserted in the source buffer, they are transmitted with a relatively low delay, however less bundles are actually inserted compared to normal ORWAR without retention classes.

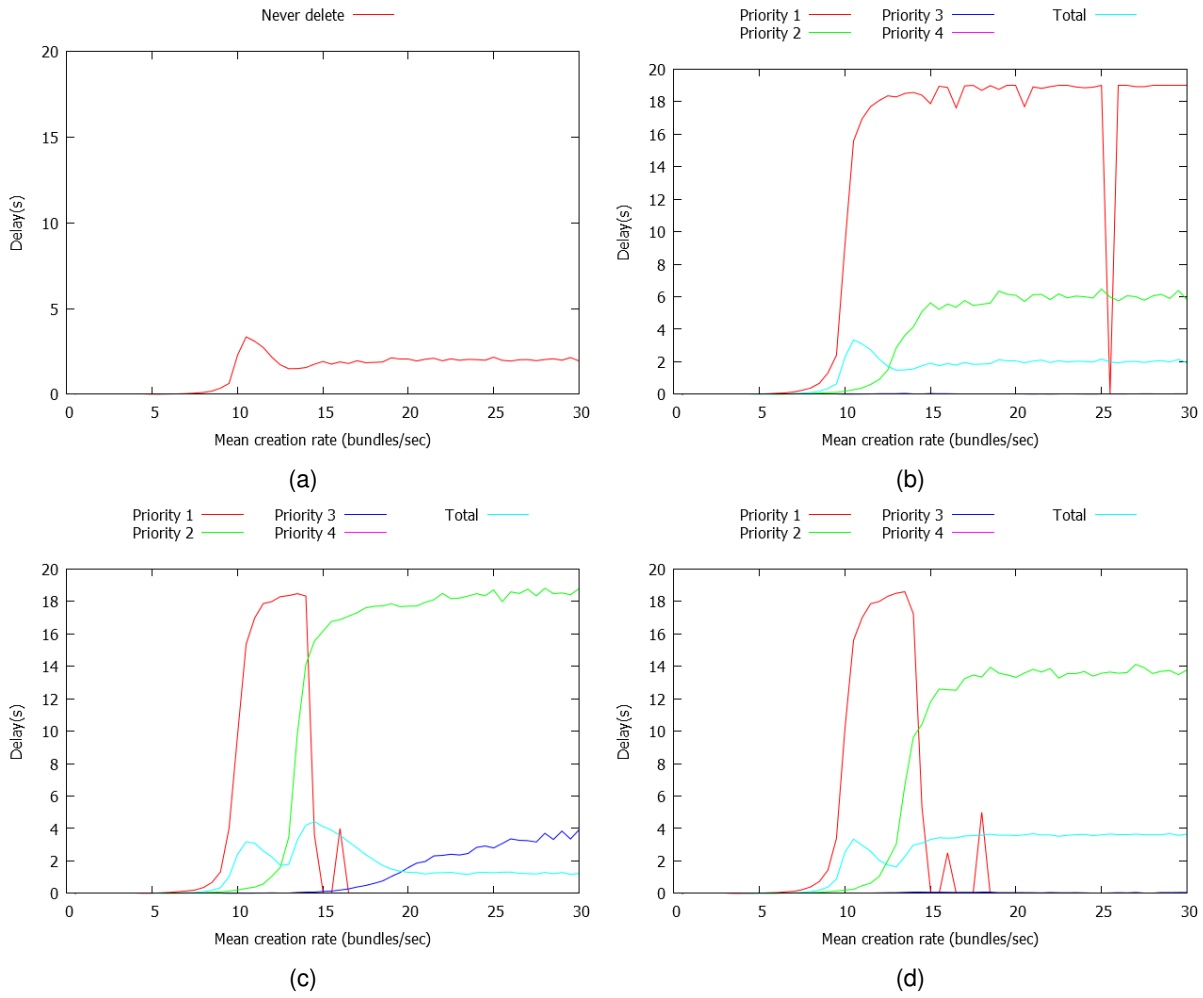
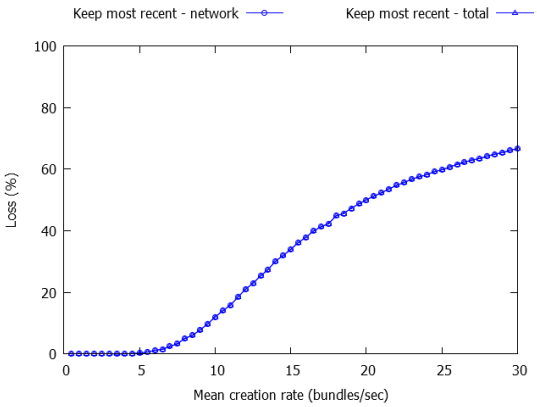
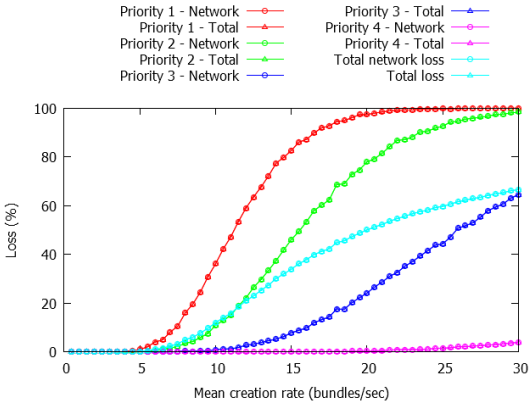


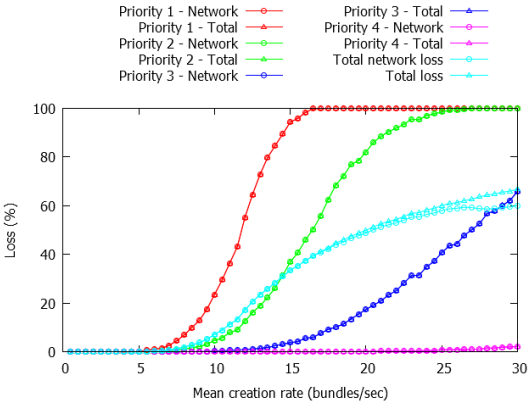
Figure 6.18: Mean delay in ORWAR with transmission priorities and constant bundle size for the stationary mobility model, using varying distributions of Never Delete: 100%(a and b), 50% (c) and 80% (d)



(a) Keep Most Recent 100%



(b) Keep Most Recent 100%



(c) Keep Most Recent 40%, Normal 60%

Figure 6.19: Loss in ORWAR with transmission priorities and constant bundle size, using the stationary mobility model and Keep Most Recent bundles

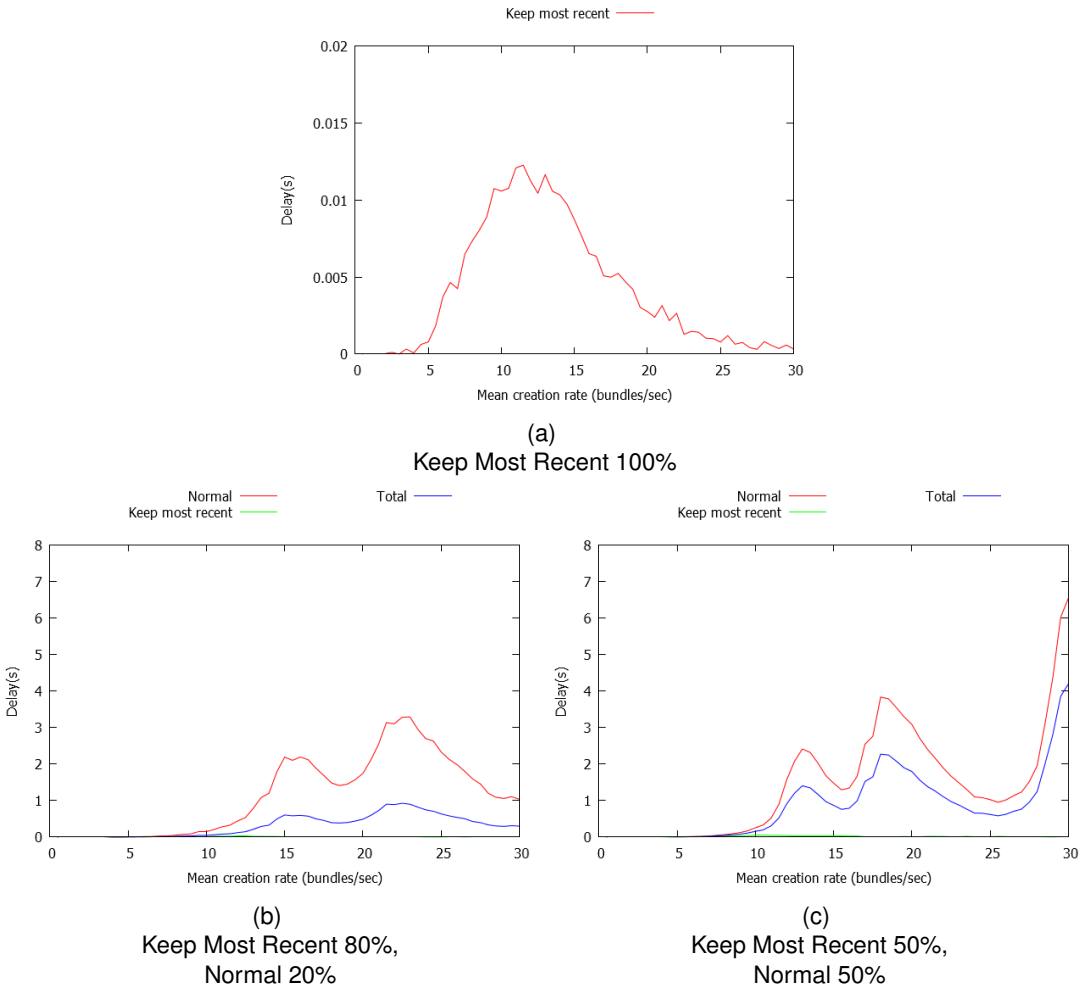
Keep Most Recent bundles in combination with Normal bundles

Next, we analyse the loss and delay for the combination of Keep Most Recent and Normal bundles. Figure 6.19 shows the loss for different percentages of Keep Most Recent traffic. In figures 6.19a and 6.19b the loss is shown when all traffic is Keep Most Recent traffic. Again we see the same values for total loss compared to Normal and Never Delete traffic when $c \geq 13$ as it is linked to the network's capacity. At this point, in the Normal and Never Delete scenarios, the total loss starts to differ from network loss. What is different in the Keep Most Recent traffic though, is that loss starts occurring at a much earlier moment. This is mainly due to the Poisson distributed nature of the generated traffic. At some times the inserted traffic is larger than the number of bundles that can be served by the network even at a $c < r_b$. When this happens, the bundles left behind may be removed by newly inserted bundles.

Similar to the case where we do not add transmission priorities, all the loss occurring in Keep Most Recent bundles is network loss. Bundles which are not immediately transmitted will likely be removed and replaced by a newer bundle before the buffer is full. Newly created bundles will therefore never be dropped before insertion.

What we also notice is that it requires a larger creation rate before low priority bundles reach 100% loss compared to the Normal and Never Delete retention classes. High priority bundles however, show loss at a smaller creation rate than in normal ORWAR. This is due to the fact that even the low priorities can remove high priority bundles if they belong to the same stream ID. Still, these high priority bundles will show loss at a higher creation rate than low priority bundles, as due to their priority they rarely stay in the buffer longer than needed. Still, due to replacing bundles with a higher priority when available, this allows lower priority bundles to arrive at the destination for higher values of c. As we can see in Figure 6.19c however, this behaviour for network loss does not require a high percentage of Keep Most Recent bundles and varies vary little as it increases.

The mean delay of bundles when we add Keep Most Recent traffic is reduced compared to normal ORWAR, as shown in Figure 6.20. The explanation for this is twofold. First of all, as we have explained in the previous section, Keep Most Recent bundles need to be delivered with low delay to their destination, otherwise they will get replaced by more recent bundles. We can see this in Figure 6.20a as the delay is extremely low when all bundles are Keep Most Recent bundles. Second, this also reduces the delay for Normal bundles. As Keep Most Recent bundles can be removed by newer versions regardless of their transmission priority, this allows Normal bundles with unaffected stream IDs to move up in the queue,



(a)
Keep Most Recent 100%

(b)
Keep Most Recent 80%,
Normal 20%

(c)
Keep Most Recent 50%,
Normal 50%

Figure 6.20: Delay in ORWAR with transmission priorities and constant bundle size, using the stationary mobility model and Keep Most Recent bundles

allowing them to be transmitted with reduced delay. It is important to note here that Keep Most Recent bundles are not inserted at the location where an obsolete bundle was removed as bundles are served in a FIFO manner within each transmission priority and one Keep Most Recent bundle may delete multiple obsolete bundles. The location where a Keep Most Recent bundle is inserted is therefore dependent on its transmission priority. The cost for this reduced delay is the increased bundle loss, even of high priority bundles.

Keep Most Recent bundles in combination with Never Delete bundles

When we analyse the combination of Keep Most Recent and Never Delete bundles, we see little difference in the aforementioned behaviour. In terms of loss, it is very similar to a distribution of Normal and Never Delete bundles, however loss of high priority bundles occurs at a lower value of c and it takes longer for low priority bundles to reach 100% loss, in accordance to the behaviour of Keep Most Recent bundles.

In terms of delay, the delay per retention priority is comparable to the combination of Keep Most Recent and Normal bundles as the Keep Most Recent bundles will also reduce the delay of Never Delete bundles in a similar way it reduces delay for Normal bundles. The delay per transmission priority on the other hand is comparable to the combination of Normal and Never Delete bundles as Priority 2 bundles will rarely suffer from starvation, even though Priority 1 bundles may still suffer from starvation due to Never Delete bundles not being removed by either a higher priority or a Keep Most Recent bundle matching its stream ID. The loss and delay for the combination of Keep Most Recent and Never Delete traffic is shown in Figure 6.21

6.2.2 Linear mobility model

Basic ORWAR performance

Analogous to Section 6.1.2, we analyse the influence of mobility on our combination of transmission priority and retention classes. When introducing linear mobility, the loss in normal ORWAR changes to

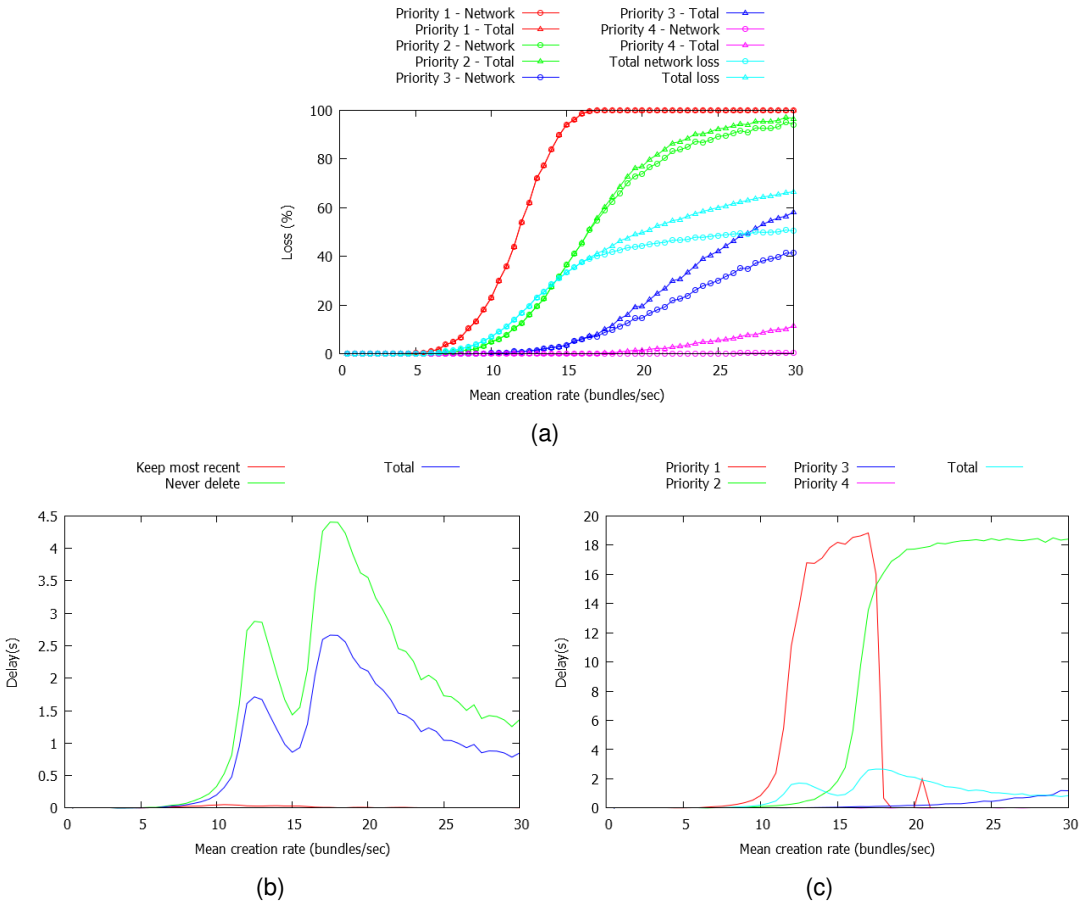


Figure 6.21: Loss and delay in ORWAR for an equal distribution of Keep Most Recent and Never Delete bundles with transmission priorities and constant bundle size, using the stationary mobility model

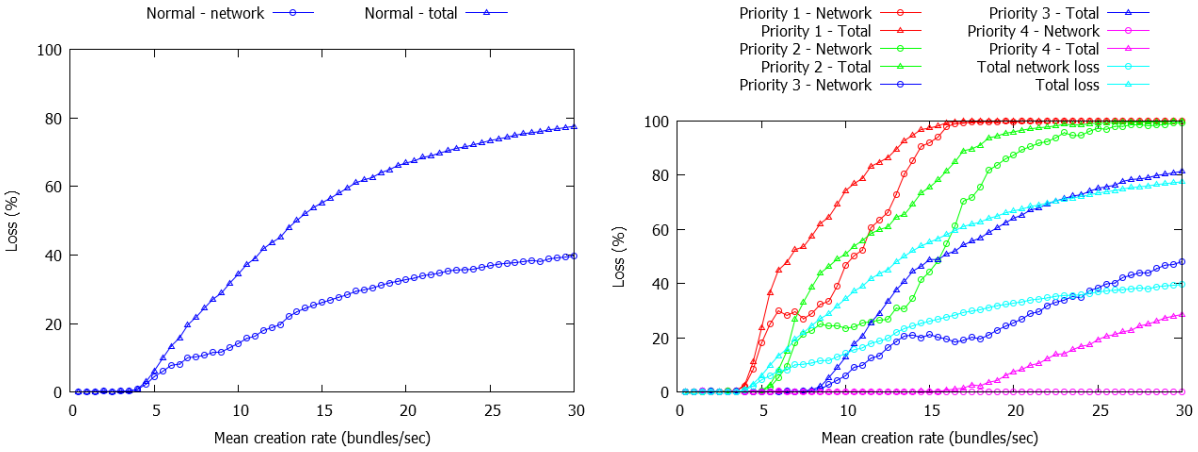
the loss shown in Figure 6.22. Immediately noticeable is the significant increase in total loss compared to Figure 6.14. The total loss in Figure 6.22 is comparable to that in Figure 6.9. Furthermore, the network loss shows different behaviour compared to Figure 6.14, with no dips in network loss showing once mobility is introduced. In addition, loss starts occurring for high priority bundles at a lower creation rate than in the stationary model.

This behaviour can be explained by the phenomenon we observed in the previous section when confronted with mobility. As node B moves out of range of node A, A's buffer will start to fill up. In addition, it will still have the bundles in its buffer it just transmitted to node B. It will have to wait for node B to return to its transmission range so it can receive acknowledgements for these bundles. Compared to the stationary model, in this model node A's buffer will fill up at a higher rate.

As soon as the buffer starts to fill up, high priority bundles will start to replace low priority bundles. However, as high priority bundles may still occupy the buffer before either their TTL expires or node B finishes its round-trip between nodes A and C, the creation rate threshold for each priority before it suffers loss will be lower. The dips in network also do not occur as most loss can be accredited to bundles being replaced by higher priority bundles or dropped before insertion rather than starvation. A bundle's TTL may expire at node A, but these bundles will generally still have been delivered by node B. This situation will only occur if the bundle's priority is too high to be replaced by newer bundles and it is still waiting on node B to return with an acknowledgement. In this case, the TTL expiring does not count as loss.

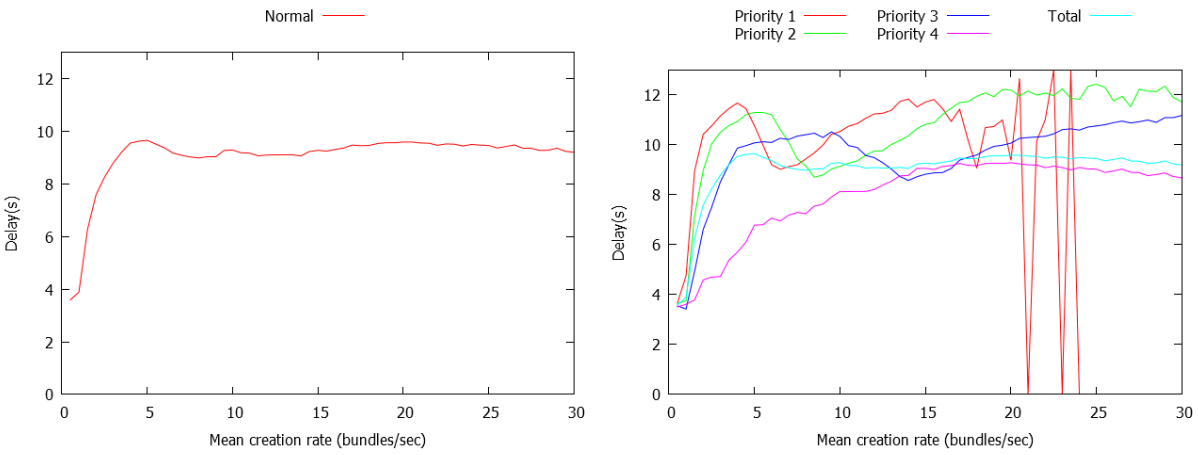
The delay for this model is shown in Figure 6.23. This delay shows different behaviour compared to the stationary model shown in Figure 6.15. In the stationary model, delay would only occur if the creation rate was high enough for loss to occur. In this case, not all generated bundles per second were able to be transmitted immediately after creation, resulting in these bundles having to wait until they could get served. This delay was exclusive for the bundles with a transmission priority suffering some loss, higher priorities would still have no delay.

Using the linear mobility model however, every bundle will suffer some delay as they wait for node B to move within transmission range and deliver them, regardless of priority. The only exception are those bundles transmitted when node B is exactly between nodes A and C, allowing end-to-end connectivity after it has just visited node C and $c < 10$. In this situation, node B's buffer will be empty as it just delivered everything to node C. At this moment, the network will temporarily behave as in the stationary mobility model. The delay in the linear model will average around 9 seconds as that is the time it takes



(a) (b)

Figure 6.22: Loss in normal ORWAR with transmission priorities and constant bundlesize using the linear mobility model



(a) (b)

Figure 6.23: Mean delay in normal ORWAR with transmission priorities and constant bundlesize using the linear mobility model

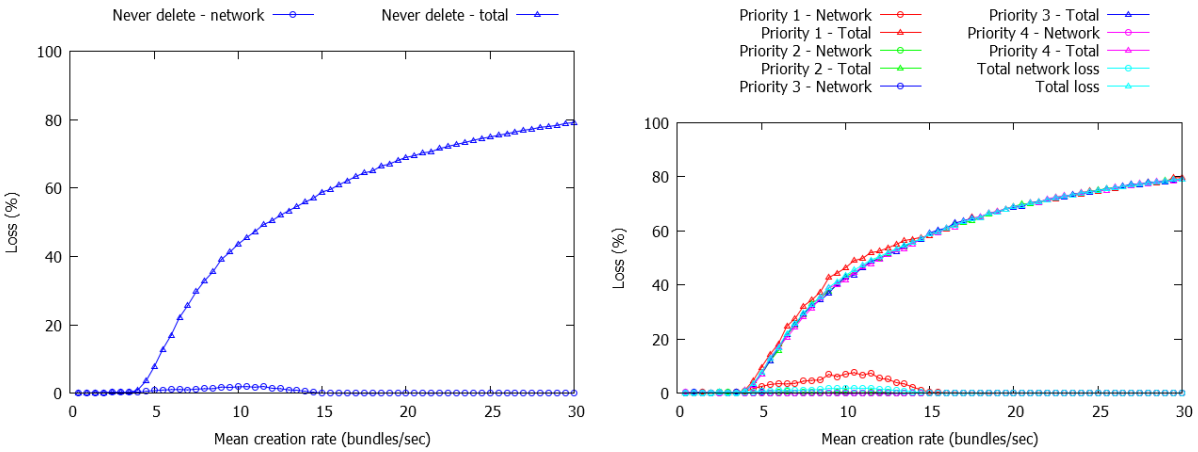


Figure 6.24: Loss in ORWAR transmitting only Never Delete bundles with transmission priorities and constant bundlesize using the linear mobility model

before node B moves out of node A's transmission range again and starts transmitting bundles to node C. Depending on the number of bundles in B's buffer as it enters node C's transmission range, it may not be able to completely empty its buffer as soon as it enters node C's range. Lower priority bundles may then suffer from additional delay as they have to wait for their turn to be transmitted to node C.

In Figure 6.23, we also notice a peak in delay for the different transmission priorities after which it will temporarily decrease before rising again. These peaks coincide with the starting occurrence of loss for these priorities. The reduced delay after this peak is caused by only recent bundles for this priority arriving as older lower-priority bundles will be replaced by newer higher-priority bundles. The valley following this peak is coinciding at the point where network loss starts to seem to be unchanged for a while. These points occur where due to the mobility of node B, the increased load does not increase network loss for that specific priority. Due to the transmission priorities being uniformly distributed, increasing c may not significantly increase the number of higher priority bundles created relative to a specific transmission priority while node B is making a round-trip. As long as no higher-priority bundles are inserted, the network loss remains relatively unchanged. This also explains why this "stable" loss level spans a larger interval of c as the transmission priority increases. However, in this situation the few lower-priority bundles that do get transmitted, will be well at the back of node B's transmission buffer, increasing their delay again as they will have to await their turn to be transmitted to node C.

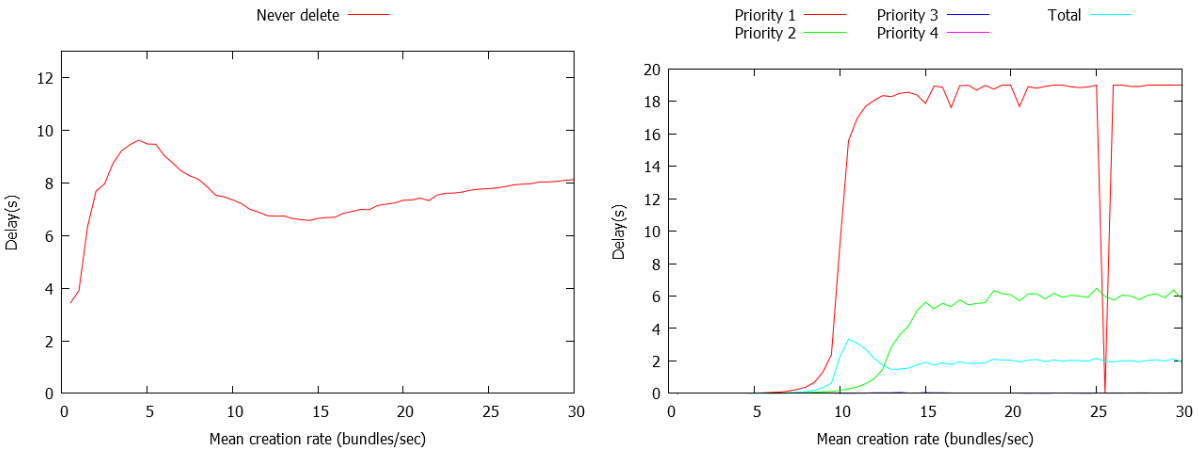
Behaviour of Never Delete bundles

For Never Delete bundles, we notice that the network delay is smaller than in the stationary model, evidenced in Figure 6.24. What is also noticeable here, is how the network loss for all priorities is lower than the stationary model with only Never Delete bundles as well. In the stationary model, bundles would be removed from node A as it received a periodic KDM-vector update. This would mean that every second, it would clear the delivered bundles from its buffer. In turn, as $c > r_b$, this would mean that while the buffer fills up, higher priority bundles would be inserted and transmitted before lower priority bundles. In the end this would lead to starvation of low-priority bundles as they would never receive a transmission opportunity.

In the linear model however, bundles will only be removed through the KDM-vector once node B finishes its round-trip between the two other nodes. When all bundles are classified as Never Delete, only the bundle's TTL and acknowledgements can remove a bundle. This means that low priority bundles also may receive a transmission opportunity as they are not constantly pre-empted by higher priority bundles as node A's buffer will have finite capacity.

We also notice there is a slight window where the network loss $> 0\%$. To explain this loss, we need to be aware of the fact that node A's buffer will still get new bundles injected as node B is out of its range, presuming there is storage space left. Loss starts occurring when node A's buffersize is too small to accommodate all bundles created during one round-trip of node B. In our simulation, node B takes 16 seconds to finish one round-trip between the nodes; 8 seconds in node A's range, and 8 in node C's. In a deterministic model, this means loss will start to occur at $c \geq 100/16 \approx 6$ bundles/sec. We see this loss occurring at an earlier moment due to our insertion rate being a Poisson process rather than deterministic.

Still, while this explains why loss occurs at all, it does not explain why network loss is occurring for Never Delete bundles. To explain this, we need to explain why at higher loads, this network loss disappears. If node A's buffer is not completely full of bundles before node B moves out of range,



(a) (b)

Figure 6.25: Mean delay in ORWAR transmitting only Never Delete bundles with transmission priorities and constant bundlesize using the linear mobility model

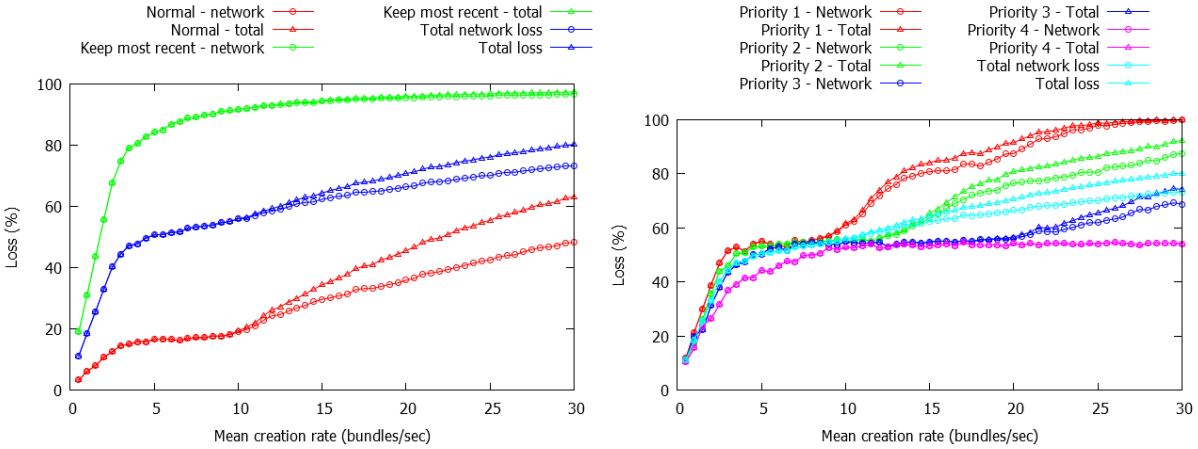
bundles will be injected which have to wait until node B moves back into range of node A and then back to node C again before they can be delivered. However, if node A's buffer is full as soon as node B moves out of range, it means that all of these bundles will be delivered as node B's buffersize is equal to that of node A. Newly created bundles will be dropped while node B is making its deliveries as the buffer is already full. As the time spent within range of node A is 8 seconds, the network loss will disappear at $c \geq 100/8 = 12.5$ bundles/sec. Again, due to using a Poisson distribution, the values from our simulation differ.

The network loss before this point is explained due to bundles having to wait longer before they can be transmitted to node C. In a worst-case scenario, it takes a bundle 16 seconds before it can be transmitted to the destination. This occurs when a bundle is inserted just as node B leaves node A's range. If c is high enough such that enough bundles can be generated to fill $> 80\%$ of node B's buffer in one round-trip in our experiments, low priority bundles may not be delivered in time. To illustrate this behaviour, presume we have a bundle b_X, with Priority 1 and injected just as node B moves out of range. It will then have to wait at least 16 seconds before it can be transmitted to node C. Our transmission rate is 20 bundles/sec, which means that if a bundle is in the last 20% of node B's buffer, it takes 5 seconds from entering node C's range until actually transmitting the bundle. If c is high enough, b_X can be in this 20% segment of the buffer as it has low priority. In this case, it will have to wait 16 seconds to get within node C's range and then wait another 5 seconds before it is transmitted by node B. However, this means the total delay will be $16 + 5 = 21$ seconds, which is larger than the TTL of this bundle, resulting in it expiring before delivery. Once bundles can only be inserted within the 8 seconds node B is in range of node A, this behaviour cannot occur anymore and all network loss disappears.

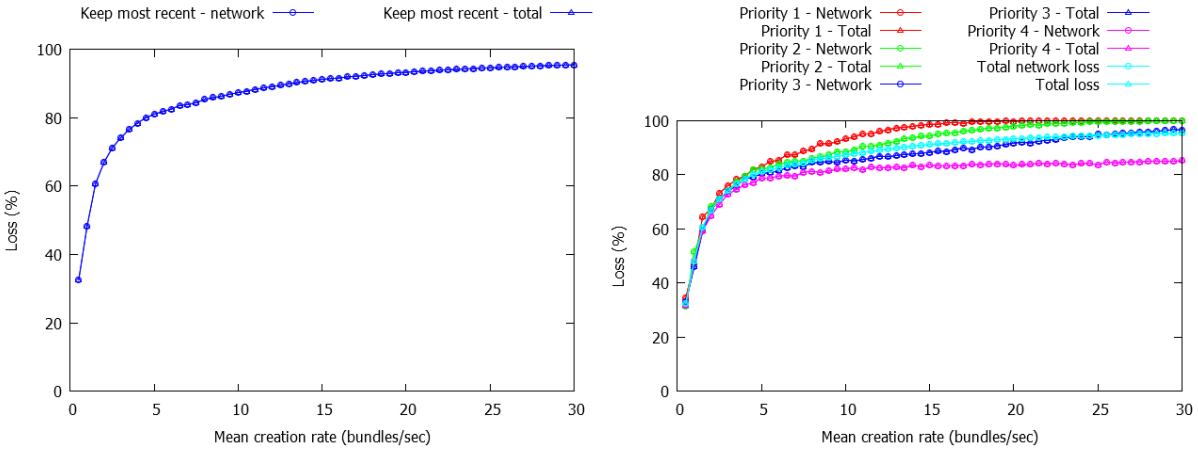
Total loss is still dependant on the network capacity and will therefore not differ from a network with only Normal classed bundles. The only difference between Normal and Never Delete in this regard, is as soon as a buffer fills up whether the newly created bundle is dropped or an already existing bundle is dropped. For lower priority bundles, the loss is more favourable when using Never Delete traffic at the cost of the loss of higher priority bundles. For higher priority bundles, 100% Normal traffic performs better in terms of both total and network loss.

The delay of Never Delete bundles in the linear mobility model also shows a different behaviour than Normal bundles, as evidenced in Figure 6.25. We again see a peak in delay here, coinciding with the point where loss starts to occur. This high delay is caused by bundles being created while node B is out of range of node A, but c is low enough so that node A's buffer is not full yet. In this case, bundles may have to wait a complete round-trip of node B before they can be transmitted to their destination. As c increases from this point onwards, the delay will get reduced again as the number of bundles showing this behaviour are reduced. In addition, as explained earlier, some bundles may expire before delivery, improving the mean delay.

Once it reaches a low point for each transmission priority, it will increase in delay again. This low point is reached as the load is of a sufficiently high value that node A's buffer will be full as node B moves out of range. At this point, as the load increases, only bundles inserted the first seconds after node B moves within node A's range will get transmitted. After these first few seconds, node A's buffer will be full and now new bundles added. Now the newly inserted bundles will have to wait longer till they enter node C's transmission range, increasing delay. This will finally result in a maximum delay of $d_A + d_C = 8 + 5 = 13$ seconds for low priority and a minimum delay of $d_A + d_C = 8 + 1 = 9$ seconds for high priority bundles. Here we declare d_A and d_C as the delays caused by having to wait within nodes A and C's transmission range respectively.



(a) Keep Most Recent 50%, Normal 50%



(b) Keep Most Recent 100%

Figure 6.26: Loss in ORWAR for different proportions of Normal and Keep Most Recent bundles with transmission priorities and constant bundlesize using the linear mobility model

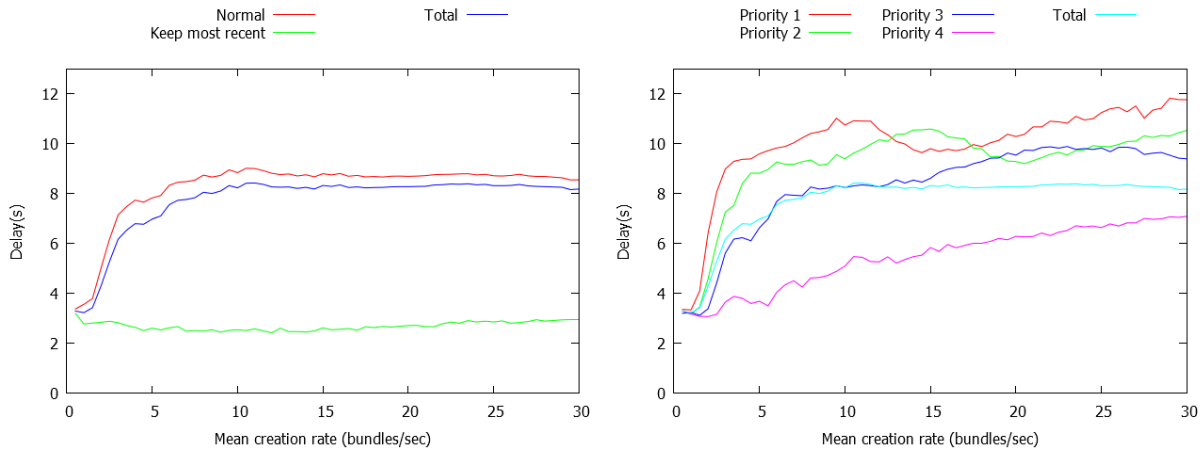
Behaviour of Keep Most Recent bundles

Finally, we observe what happens when we increase the number of Keep Most Recent bundles relative to Normal bundles. The loss shows some interesting behaviour, shown in Figure 6.26. As we increase the percentage of Keep Most Recent bundles created, we notice that loss for all priorities increases rapidly until it reaches a stable value for the highest transmission priority. At this point onwards we also start observing a difference between total and network loss. Furthermore, the behaviour becomes similar to using only Normal bundles, although all priorities have a slower increase in loss as c increases compared to transmitting only Normal bundles.

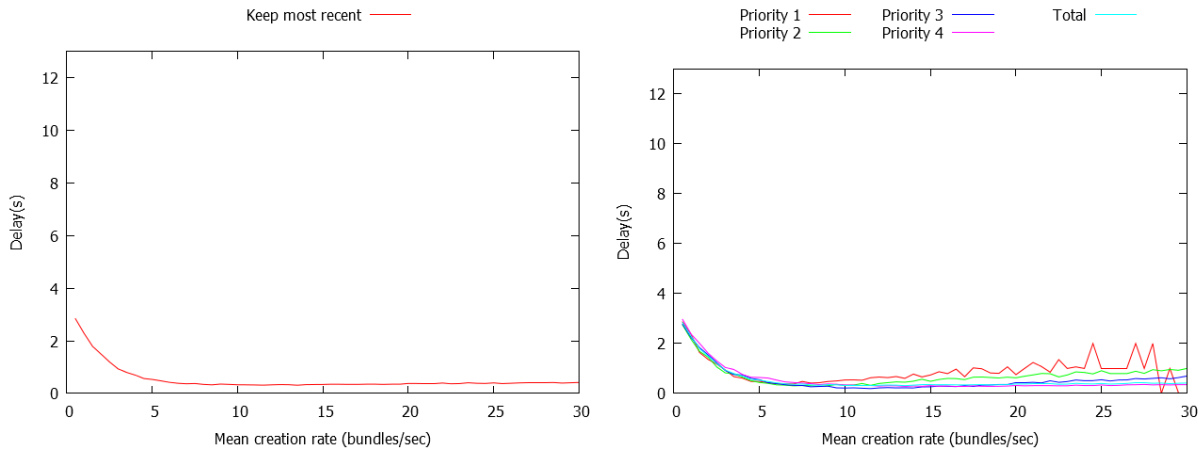
This behaviour does not occur in the stationary model as bundles are transmitted end-to-end in said model. When bundles are transmitted end-to-end, loss only occurs at the source node as lower priority bundles have to wait until they are served. As the proportion of Keep Most Recent bundles increases, bundles that are not immediately transmitted are at increased risk of being replaced by newer bundles, regardless of transmission priority.

When we introduce mobility however, bundles will not be transmitted end-to-end and therefore even transmitted bundles are not immune to being replaced by a newer bundle. While node `B` is still only within node `A`'s transmission range, bundles in both nodes can be replaced by newer copies as node `B` has not had the opportunity yet to transfer its buffer contents to node `C`. For a high enough c, this means that only the last created bundles before node `B` moves out of node `A`'s range will be delivered to `C`. Of course, this is only valid for bundles which have a stream ID that allows Keep Most Recent bundles. This will start happening at a relatively low mean creation rate as the proportion of Keep Most Recent bundles increases, as there are only four distinct stream IDs which allow Keep Most Recent bundles. Similarly, as soon as node `B` leaves node `A`'s transmission range, newly inserted bundles are at a high risk of being replaced as there is no node to transfer them to. This results in a near 100% loss for all transmission priorities as the proportion of Keep Most Recent bundles reaches 100%.

After this stable loss level is reached, the additional loss behaviour is caused by the Normal bundles which do not have a stream ID which can contain Keep Most Recent bundles. These bundles follow the same loss patterns as traffic using only Normal bundles, based on the network capacity. The reason why their loss increase is slower than when transmitting only Normal bundles is that the Keep Most



(a) Keep Most Recent 50%, Normal 50%



(b) Keep Most Recent 100%

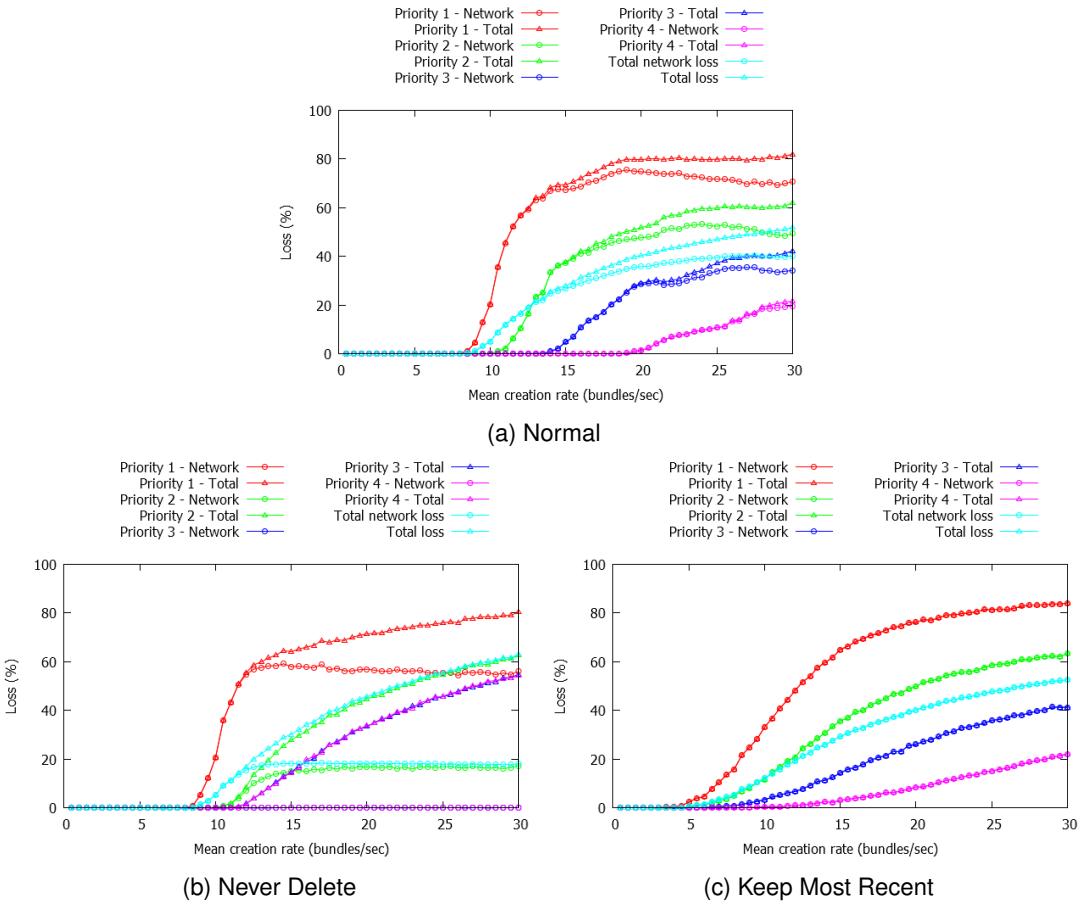
Figure 6.27: Mean delay in ORWAR for different proportions of Normal and Keep Most Recent bundles with transmission priorities and constant bundlesize using the linear mobility model

Recent bundles increase the time necessary for a buffer to completely fill up by periodically removing older bundles. It will therefore take longer before lower priority bundles will need to be replaced by higher priority ones. Furthermore, it reduces the risk of a bundle expiring before it can be delivered by moving it forward in the buffer by deleting obsolete bundles before it.

As we look at the delay in this scenario, shown in Figure 6.27, we notice similar behaviour as to when no transmission priorities are used. In general, the mean delay is less than using only Normal bundles, at a cost of increased loss. Also, Figure 6.27b backs up the earlier claim that only the bundles last inserted as node B moves out of range of node A get transmitted, even at a relatively low c. This is evidenced by the very low delay for all bundles, generally between 0 and 1 second, while in a normal scenario this delay is significantly larger as bundles have to wait for node B to return. In addition we also see that transmission priority has little to no influence on this metric in this scenario.

Once Normal bundles are also added to the mix, shown in Figure 6.27a, we still see a lower mean delay for all bundles. The difference in delay between transmission priorities is caused by Normal bundles which do not have a stream ID which can contain Keep Most Recent bundles. The delay for all priorities is still lower as bundles move to the front of the buffer as obsolete bundles are removed. In addition, the buffers of nodes A and B will not be as full as when using only Normal bundles further decreasing delay.

We again do not discuss the non-linear mobility model in this subsection as its performance is similar to the linear model. While the utility-per-bit value is not equal for every bundle as we add transmission priorities, we still have a constant bundlesize. This means that regardless of a proper contact window estimation, we can still only transmit a fixed number of bundles every contact window. In the next subsection we also vary the individual bundlesizes which means a proper contact window estimation is required to efficiently transfer data. Attempting to transmit a bundle with too high a bundlesize relative to the remaining contact window will be detrimental to performance. Only when proper contact window estimation is required will the performance differ between the linear and non-linear model.



(a) Normal

(b) Never Delete

(c) Keep Most Recent

Figure 6.28: Loss per transmission priority in ORWAR for different retention priorities using the stationary mobility model

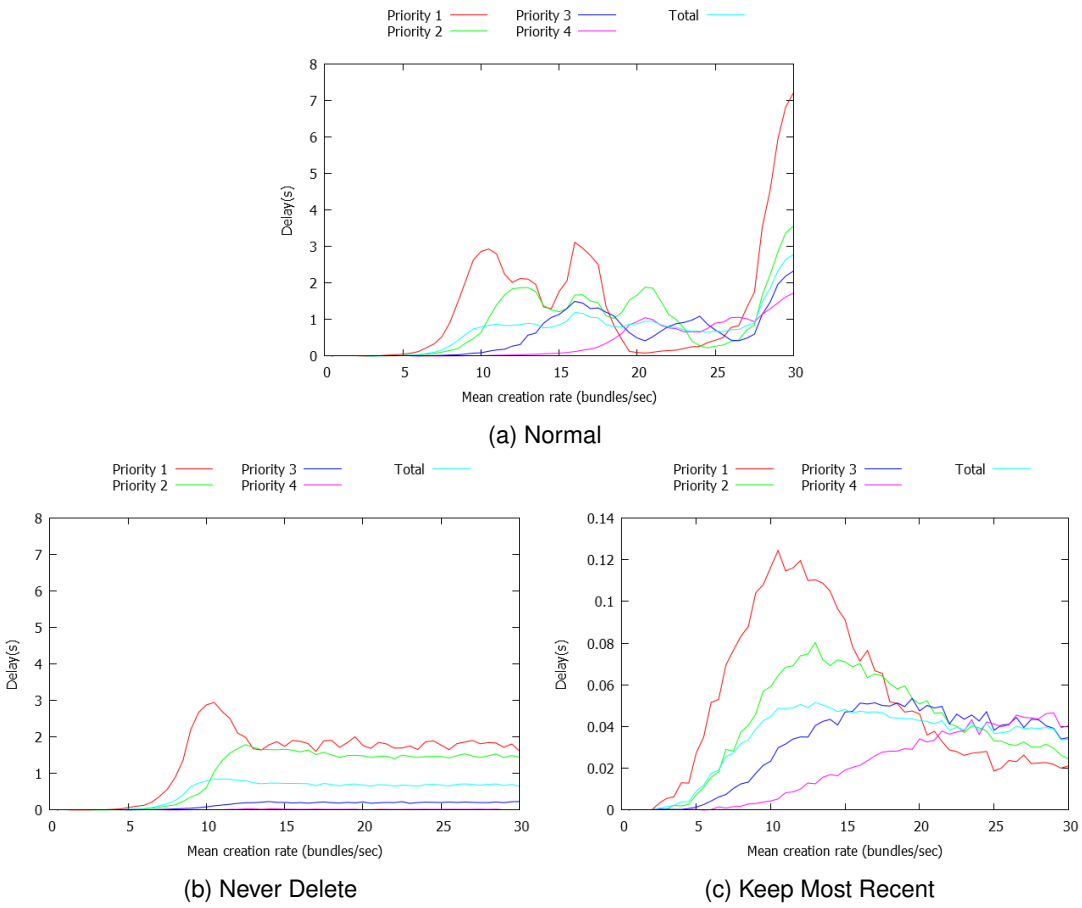
6.3 Retention classes with transmission priorities and varying sizes

For our final simulations using the simplified mobility models, we also vary the bundlesizes. This means that for 100% Normal traffic, behaviour is similar to normal ORWAR, including the utility-per-bit scheduling of bundles. As we now introduce different bundlesizes, we may not make optimal use of the available contact window anymore. Previously our bundles had a constant size of 5 bytes; with a transmission capacity of either 10 or 20 bundles/sec, this means there was no risk of incomplete bundle transmissions or idle time as no bundles would fit in the remaining time. Furthermore, in this situation the estimation of contact window actually matters as it will determine whether bundles have to wait or will be transmitted. In these experiments we therefore analyse the difference in performance of ORWAR with and without retention classes in a predictable environment. Using the results from this analysis, we attempt to provide some indication of an optimum distribution of retention classes on the network for use in the realistic mobility model.

In the stationary model, the contact window is ∞. The scheduling of which bundles to transmit is therefore similar to the situation where bundlesize is equal for all bundles, though bundles are ordered by utility-per-bit rather than only their transmission priority. The results of this scheduling can be seen in Figure 6.28 for the three different retention classes.

We notice that the behaviour is similar to the case where we used a constant bundlesize. The main differences we see here is that the average loss is lower and no transmission priority suffers from 100% loss. This lack of 100% loss is caused by the fact that even low priority bundles may be transmitted if its size is small enough, rather than expiring or getting replaced by a higher priority and/or more recent bundle. Still, these lower priority bundles will still suffer from loss at a lower c than higher priorities as the range of utility-per-bit values a low priority bundle can attain is lower than that of a high-priority bundle. Therefore, the majority of these low-priority bundles will still end up at the rear of the buffer.

The delay however shows different behaviour when adding different bundlesizes. This is most significant for Normal bundles, where the delay is not only significantly reduced but also shows completely different behaviour, evidenced in Figure 6.29. This is easily explained by the fact that lower priority bundles do not suffer from starvation by definition. Even low priority bundles may be transmitted with a low delay if their size is small enough. When using equal bundlesizes, the delay of the lowest priority would be maximal, while delay of higher priorities was kept to a minimum, until its loss reached 100%. At this point, the loss of the next highest priority would increase along with its delay. Now there still is a



(a) Normal

(b) Never Delete

(c) Keep Most Recent

Figure 6.29: Mean delay in ORWAR for different retention classes, using the stationary mobility model

difference in delay between low and high priorities, yet the difference is not that big anymore, with even high priority bundles able to suffer from delay at lower creation rates.

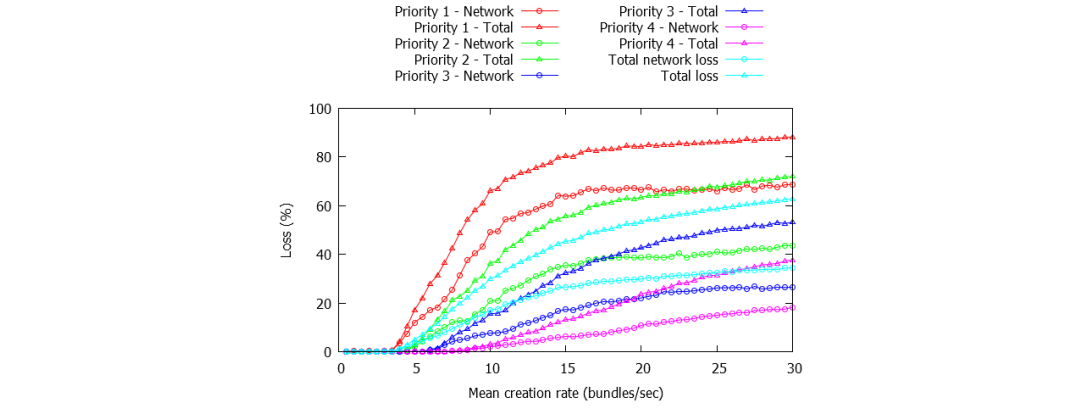
When observing the loss of ORWAR using the linear mobility model, we again notice that the behaviour is very similar to the behaviour when using constant bundlesizes. Again, similar to the stationary model, we see that there is no 100% loss for any transmission priority. Another difference compared to the constant bundlesize scenario we notice, is that loss occurs for Priority 4 bundles at a much lower creation rate. We already stated that due to node B's mobility, loss occurs at a lower creation rate compared to the stationary model as the buffer fills up. Due to the varying bundlesizes, Priority 4 bundles may still end up in the rear of the buffer if their size is large enough. These bundles will then be replaced by bundles with a more favourable utility-per-bit until bundles can be transmitted to node B again.

Furthermore, when we compare the loss of Never Delete and Normal bundles compared to the scenario using constant bundlesizes, we observe a significant reduction in loss. For Keep Most Recent bundles however, the loss reduction is less significant. This is due to different bundlesizes not affecting the fact that only the bundles inserted just before node B leaves node A's range have a chance at arriving at the destination.

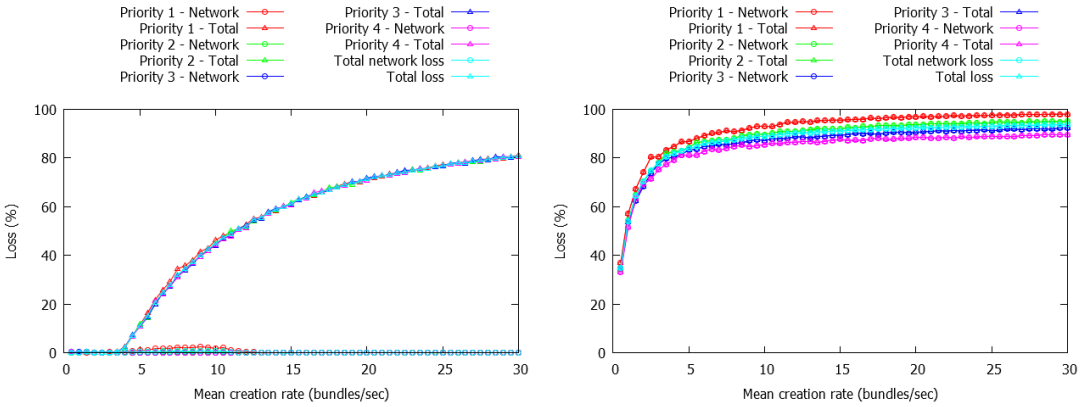
Similarly, the delay when introducing varying bundlesizes differs very little from constant bundlesizes. The main factor for delay in the linear mobility model is the travelling time of node B between nodes A and C. Transmissions from source to destination are still done in bursts, with node A first transmitting its buffer contents to node B after which node B will transmit these contents to node C. The average delay of each burst is equal, dependent on the round-trip time of node B. Only for Never Delete bundles we see a slight decrease in delay. This is likely caused by newly inserted bundles not necessarily being inserted at the back of the buffer based on their relative transmission priority, increasing their delay.

When comparing the linear and non-linear mobility model, shown in Figure 6.32, we actually notice little to no difference compared to the linear mobility model. We expected to see increased loss as ORWAR reverts to best-effort transmission due to incorrectly estimating the contact window. Due to this best-effort transmission, some bundles may suffer from loss as the contact window ends mid-transmission. This however does not seem to be the case in this scenario.

One possible explanation for this is we chose our bundlesize to be smaller than the transmission rate of the network. This would decrease the number of incomplete transmissions relative to all transmissions due to a wrong estimation of contact window. For every transmission that may be interrupted, there is at least one other that was correctly transmitted. Increasing the bundlesize will also increase the risk of incomplete transmissions. In addition, the actual contact window with node B relative to the



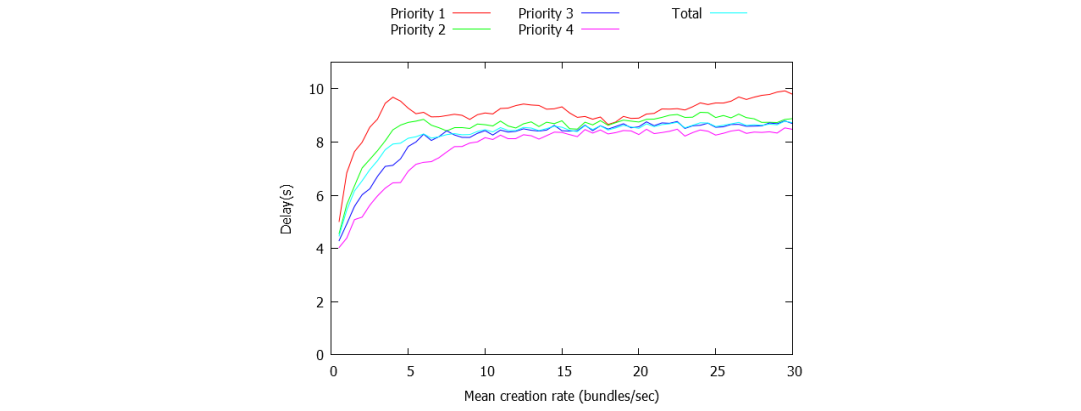
(a) Normal



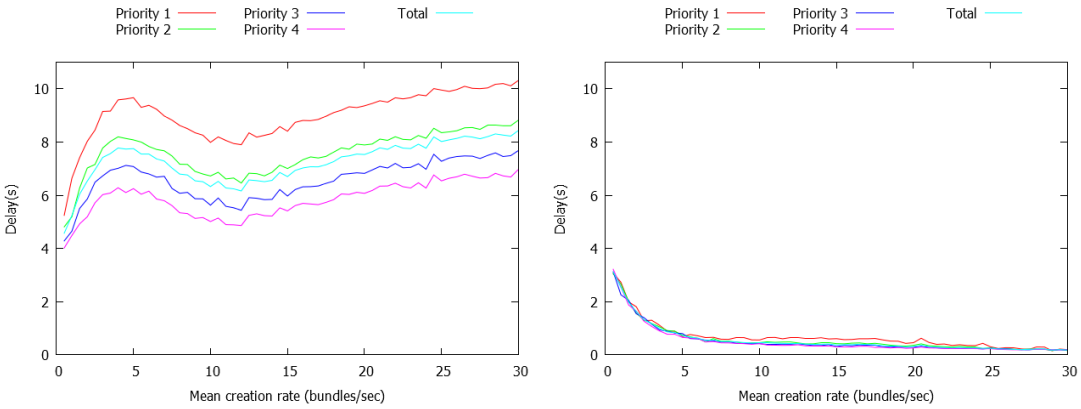
(b) Never Delete

(c) Keep Most Recent

Figure 6.30: Loss per transmission priority in ORWAR for different retention priorities using the linear mobility model



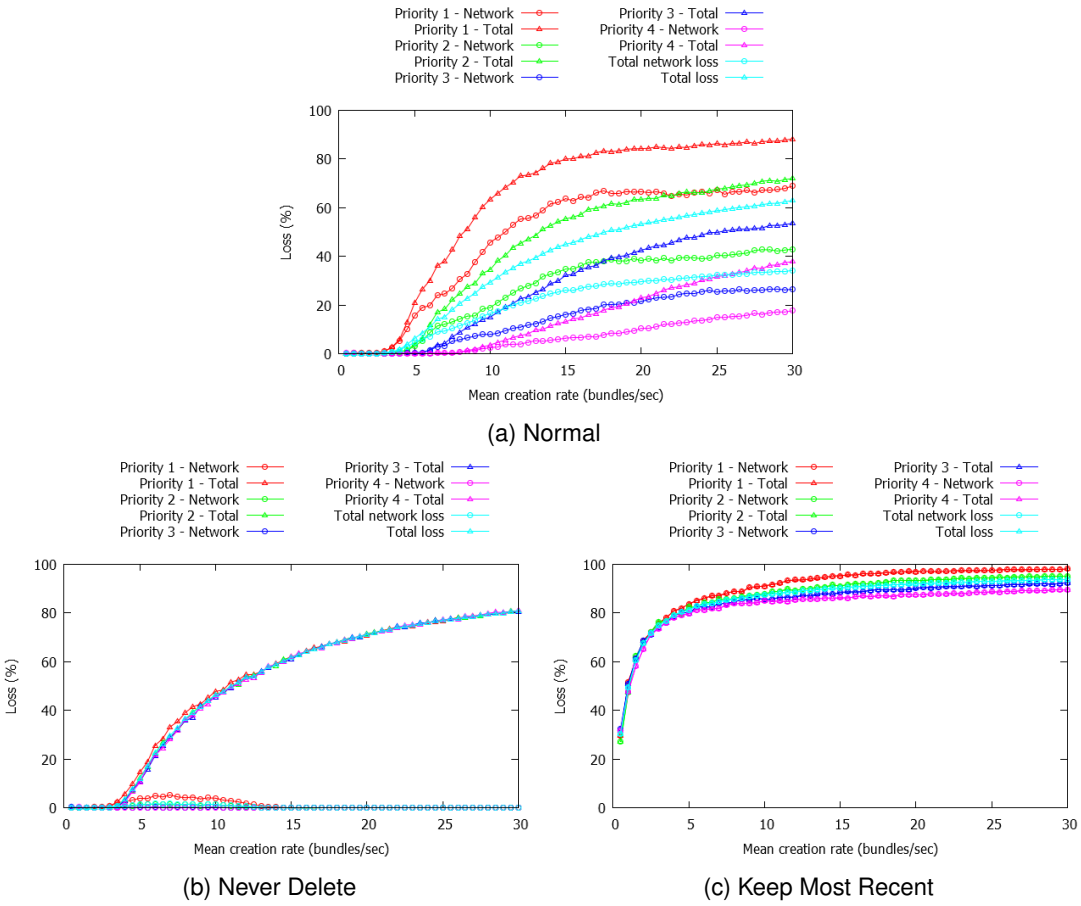
(a) Normal



(b) Never Delete

(c) Keep Most Recent

Figure 6.31: Mean delay per transmission priority in ORWAR for different retention priorities using the linear mobility model



(a) Normal

(b) Never Delete

(c) Keep Most Recent

Figure 6.32: Loss in ORWAR for different retention classes using the non-linear mobility model

transmission rate is large enough for a node to completely transmit its buffer contents during the contact window. The buffersize of all nodes is 500 bytes, furthermore we have a transmission capacity of 100 bytes/second. During one contact opportunity, we can transmit $8 * 100 = 800$ bytes, which is more than enough to transmit the complete buffer. Therefore there is barely any difference between best-effort mode and an accurate contact-window estimation. For reduced performance to occur, nodes `A` or `B` should not be able to transmit its complete buffer contents during one contact opportunity.

6.4 Summary

In this section we have analysed multiple traffic scenarios using different mobility models. By first removing the transmission priorities and keeping an equal bundlesize, we were able to determine some characteristics of ORWAR and our retention classes. By adding the transmission priorities first and then also varying the bundlesize, we were able to analyse the impact of our retention classes on ORWAR's performance.

First of all we found that that ORWAR needs to periodically exchange KDM-vectors in case nodes are stationary or have very long contact windows. If this is not done, bundles will be kept in the source buffer for too long and newer bundles may not be inserted onto the network.

We also notice the importance of including transmission priorities in combination with our retention classes. Not only do transmission priorities reduce the mean bundle delay, but without transmission priorities, there will be no distinction in performance between Never Delete and Normal bundles. Subsequently, having a variable bundlesize will aid in the scheduling of bundles, reducing the probability of total starvation of low priority traffic.

When using Never Delete bundles, having too high a percentage of Never Delete traffic with regards to total traffic will reduce your storage efficiency and increase loss as bundles are not inserted on the network due to storage buffers reaching maximum capacity. However, the loss suffered by bundles classed as Never Delete which are actually inserted on the network is kept to a minimum.

Keep Most Recent traffic on the other hand, suffers large loss but offers greatly reduced delay for all traffic. Through the use of stream IDs, bundles can be divided into two groups: those that are, and those that are not affected by Keep Most Recent bundles. For the former group, delay is reduced as waiting in a buffer for too long increases the probability of being deleted by a more recent Keep Most Recent bundle. This ensures that only those bundles which are propagated fast enough will eventually

reach their destination. The latter has their delay reduced as the deleting of obsolete bundles reduces the waiting time of all bundles in a buffer, reducing delay. Still, the percentage of total traffic classed as Keep Most Recent traffic should not be too high as values greater than 50% will offer no significant improvements in delay, but greatly increase loss. This threshold value may change dependent on the stream ID distribution.

The mobility of nodes also plays a key factor in ORWAR's performance. By simulating intermittent connectivity between nodes, we have shown the need for an acknowledgement retention time to not only make bundle transmissions delay-tolerant, but acknowledgement transmissions as well. Similarly, bundles may occupy a node's buffer for long periods of time while they wait for an acknowledgement to be received. This occupation of buffer space, while not transmitting, can result in the non-injection of newly created bundles on to the network. Using transmission priorities in combination with varying bundle sizes however can alleviate this issue as less important bundles are removed to free storage space.

The performance of ORWAR, with or without retention classes, is dependent on the following key aspects:

- Use of transmission priorities and varying bundle sizes
- Storage buffer size
- TTL of bundles, and
- Mobility of nodes (servicing times and travel patterns)

Lastly, we have determined that the contact window need not always be calculated accurately. This depends on the average bundlesize and the calculated contact window. Should the average bundlesize be significantly smaller than the total amount of data which can be faultlessly transmitted during a contact window, an inaccurate estimation will only have minor influence on performance. Having a bundle size too small however, may limit the amount of goodput on a network due to increased overhead.

Chapter 7

Results using the realistic mobility model

In the previous section we analysed the behaviour of ORWAR and the influence of our different retention classes through the use of some simplified mobility models. This section deals with the performance of ORWAR with and without retention classes in a more realistic mobility model as described in Section 5.1. The results gained in this section are reliant on too many variables to fully describe the behaviour. Instead, we compare the performance of normal ORWAR to our version with the implemented retention classes.

7.1 Retention class parameters

First of all, we need to determine the retention class parameters for which to test our QoS-enabled ORWAR. In the previous section, we have observed how the different retention classes behave and influence each other. However, we have only compared two classes at a time, not all three at once. Aside from network performance metrics, we should also take storage efficiency into account. This leads us to determine that we should not have too high a proportion of Never Delete bundles as they can cause a buffer to fill up rapidly by not replacing bundles with more important or more up-to-date bundles. As the required TTL in military networks is at least 24 hours, this poses a great risk. Should a buffer fill up with Never Delete bundles, it prevents newly created bundles from being injected into the network. This could have dire consequences as high priority messages may not be injected.

Never Delete bundles improve performance for low-priority bundles in a non-stationary network at the cost of high priority bundles. We therefore propose the Never Delete retention class is only adopted by Priority 1 and 2 bundles. High priority bundles are already at reduced risk of being removed due to their position in the buffer. This in turn also reduces the total number of Never Delete bundles on the network by limiting the bundles it can apply to. By not applying Never Delete to high priority bundles we also reduce the risk of high priority bundles not being injected.

Keep Most Recent bundles on the other hand, would increase loss of all priorities due to deleting of obsolete bundles, though would also ensure severely reduced delay. This behaviour is generally favoured for high priority, near real-time communications. Keep Most Recent classes are best applied to bundles which will be transmitted in the near future as waiting in a node's buffer increases the probabilities of it being replaced by a newer version. Applying this to low-priority traffic is therefore sub-optimal. A low priority bundle with the Keep Most Recent retention class can delete high-priority traffic, but has no guarantee it will be transmitted soon itself. In addition to being deleted by a newer Keep Most Recent bundle, a low-priority Keep Most Recent bundle is also at risk of being deleted by any higher priority bundle. This may result in data unnecessarily getting lost on the network. We therefore propose that the Keep Most Recent class is only applied to Priority 3 and 4 bundles.

The major advantage of Keep Most Recent bundles is its delay reduction. In the simplified mobility models we found the delay does not change significantly anymore as the proportion of Keep Most Recent traffic $\geq 50\%$. However, increasing Keep Most Recent traffic beyond 50% does increase loss. Therefore, Keep Most Recent traffic be $\leq 50\%$ of the total network traffic. Higher values will delete too many bundles before they can be propagated over the network.

These analyses result us to determine that for optimum efficiency using these retention classes, the majority of all traffic should still be classed Normal, with Keep Most Recent traffic being $\leq 50\%$ and Never Delete traffic having a significant minority share of the traffic based on network load and buffersizes. Furthermore, the Normal class can be used for all four transmission priorities, while Keep Most Recent is used solely for high-priority traffic and Never Delete for low-priority traffic. As transmission priority is not independent of retention class anymore, this results in the transmission classes not necessarily being uniform distributed based on the proportions of Keep Most Recent and Never Delete traffic. If the

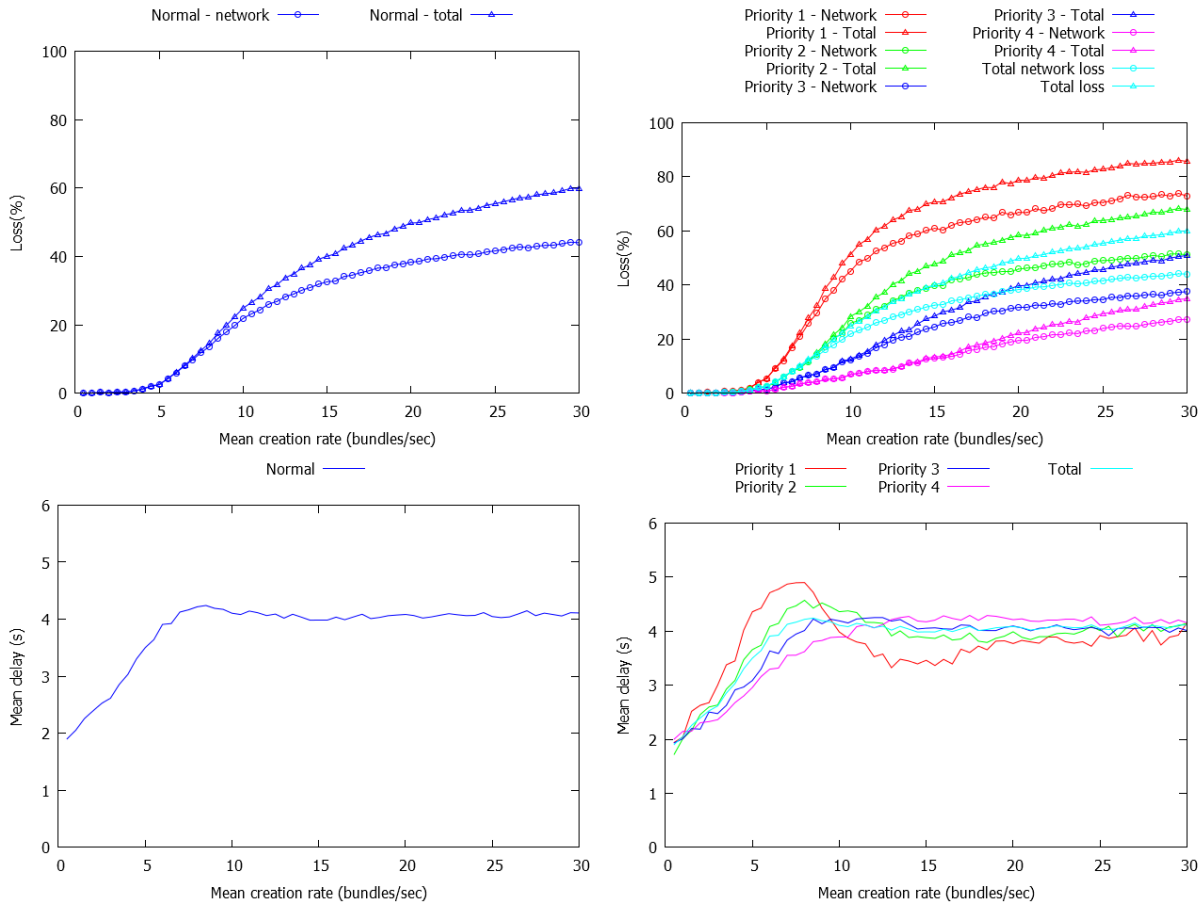


Figure 7.1: Loss and mean delay in ORWAR using the realistic mobility model

proportion of Keep Most Recent traffic is unequal to the Never Delete traffic, transmission priorities are not uniform distributed over the network anymore. Within each individual retention class, transmission priorities are still uniform distributed. In this thesis we want to analyse the influence of retention classes on performance, therefore we do not force a network-wide uniform distribution of transmission priorities.

7.2 Basic ORWAR performance

For reference, we first run the realistic model for normal ORWAR without different retention classes. The results of this simulation can be seen in Figure 7.1. As expected, the behaviour shown here is similar to that of figures 6.30a and 6.31a. There are two main differences between these two. First, the creation rate at which the difference between network and total loss starts to increase. Second, the mean delay is significantly lower in the realistic model.

Regarding the loss, we see that while the total loss in both models differ very little, the network loss in the realistic model is significantly higher. In the realistic model, bundles can be inserted at any node of the network and can have any other node in the network as its destination. The mean creation rate is the creation rate *per node*, this means that while in previous experiments the creation rate was also the total number of bundles created on the network, now this is not the case anymore. The number of bundles created on the network is the creation rate multiplied by the number of nodes on the network. The total loss is still relatively unchanged as this is dependent on the network capacity. Network loss will increase however, as with more bundles on the network, the probability to be replaced by a more important or more recent bundle increases. In addition, when nodes meet, both nodes may already have a number of bundles in the buffer, meaning there may not be enough space for all bundles to be transferred from one node to another. Combined with the more random movement patterns also increases the number of bundles which are removed due to their TTL expiring.

In regards to the delay, even though we notice that the mean delay is lower in our realistic model, we also notice that the highest priority bundles also seem to have the highest delay as the creation rate increases, especially as $c \geq 13$. This behaviour is caused by only those low-priority bundles which had a low delay actually being delivered while others are lost. For a low priority bundle to be delivered in the original version of ORWAR, it has to either have a small size or does not have to traverse multiple hops to its destination. Either of these two scenarios ensures the bundle is relatively at the front of the transmission buffer, either due to a favourable utility-per-bit or the next hop being the destination.

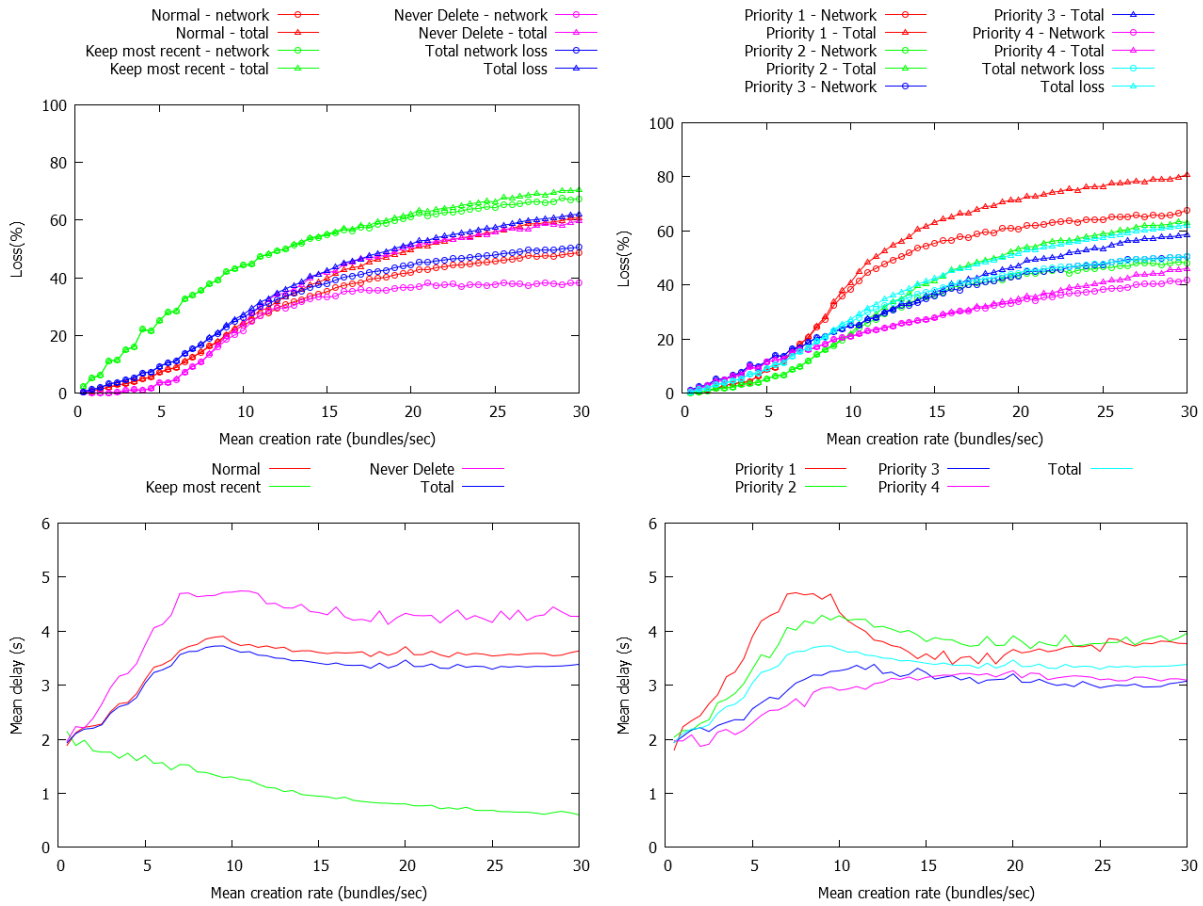


Figure 7.2: Loss and mean delay in ORWAR using the realistic mobility model
70% Normal, 15% Keep Most Recent, 15% Never Delete

7.3 QoS-enabled ORWAR performance

As we have determined the relative proportions between the different retention classes, we repeat this simulation but now add the retention classes using different distributions. We set the number of Normal bundles to be 70% as we determined it has to still hold the majority share of bundles, but use different proportions for the other two classes. The results from these experiments are shown in Figures 7.2, 7.3 and 7.4.

In terms of loss, we notice little difference between these three, though loss values seem to be slightly favourable when Keep Most Recent traffic outweighs Never Delete traffic. In this case, the total loss is lower even though network loss is slightly increased. As the difference between network and total loss is determined by bundles which are created but not injected, we find a lower total loss favourable over lower network loss. However, in all cases, the differences are minimal.

When Keep Most Recent bundles outweigh the Never Delete bundles however, we also notice that the loss of Normal and Never Delete bundles has a lower value than the total loss over all retention classes. These are the two retention classes we also prefer reduced loss for. Loss in Keep Most Recent traffic is accepted as the loss is required to transmit more recent versions of a bundle.

Furthermore, when comparing these to normal ORWAR's performance, we notice that network loss is larger compared to normal ORWAR. This is understandable as with the Keep Most Recent class behaviour, bundles are more often deleted instead of only when a node's buffer is full. Furthermore, the loss of Priority 1 bundles is lower when adding retention classes at the cost of increasing the loss for Priority 4 bundles. Priorities 2 and 3 are now closer to the total loss on the network. In effect, the difference between transmission classes in terms of loss has been reduced, with every class having been moved closer to the loss measured over all traffic.

In terms of delay, we again notice little difference between the three distributions, though again Keep Most Recent outweighing Never Delete bundles offers favourable delay metrics, even though it is a small improvement. All three metrics however, offer a reduced mean delay compared to normal ORWAR. Again this is explained due to the Keep Most Recent class behaviour, it is possible for a single bundle to delete many other bundles simply because it is newer, even if the buffer is not full. Only Never Delete bundles are immune to this behaviour, however only a minority of the bundles are assigned this retention class. We have already shown that this pre-emptive deleting of bundles will reduce delay both by reducing the waiting time of a bundle within a buffer, as well as ensuring only those bundles that are

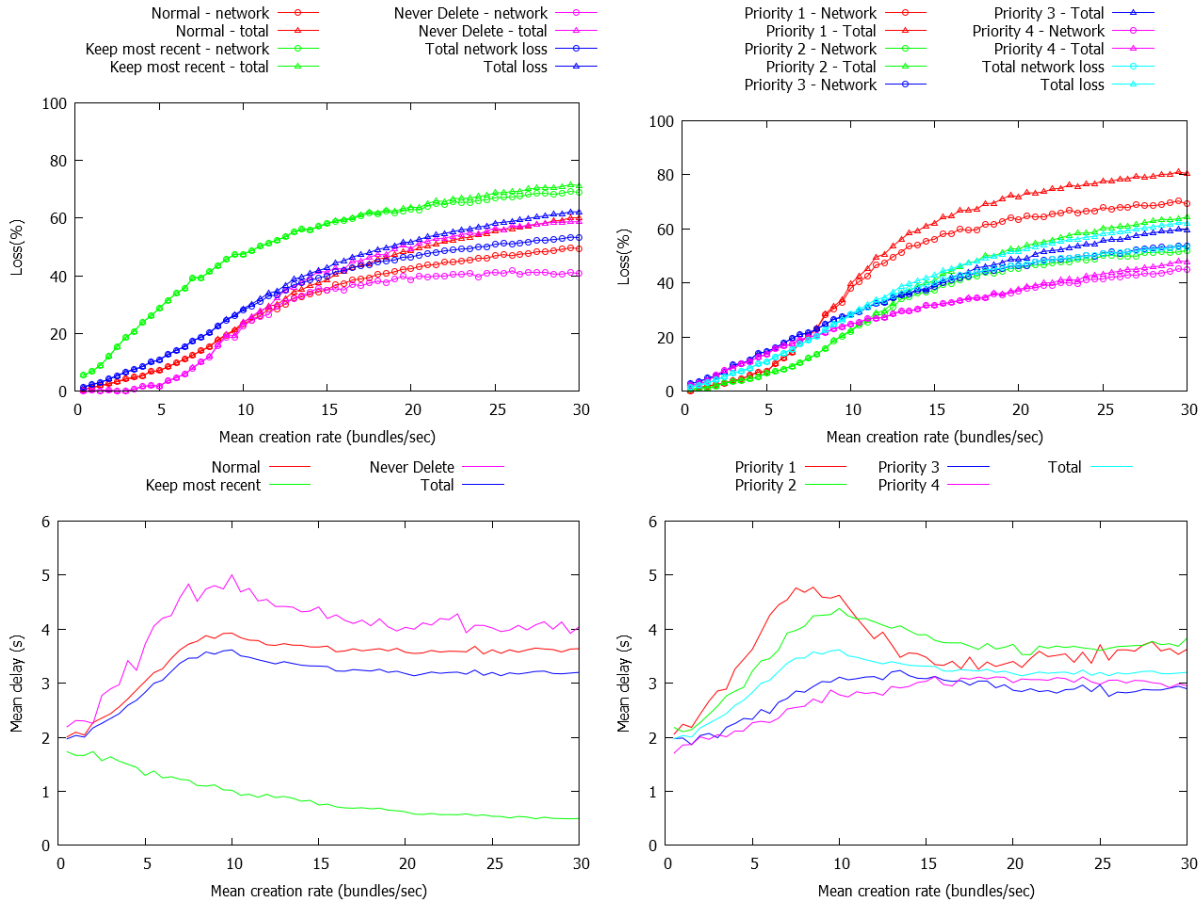


Figure 7.3: Loss and mean delay in ORWAR using the realistic mobility model
70% Normal, 20% Keep Most Recent, 10% Never Delete

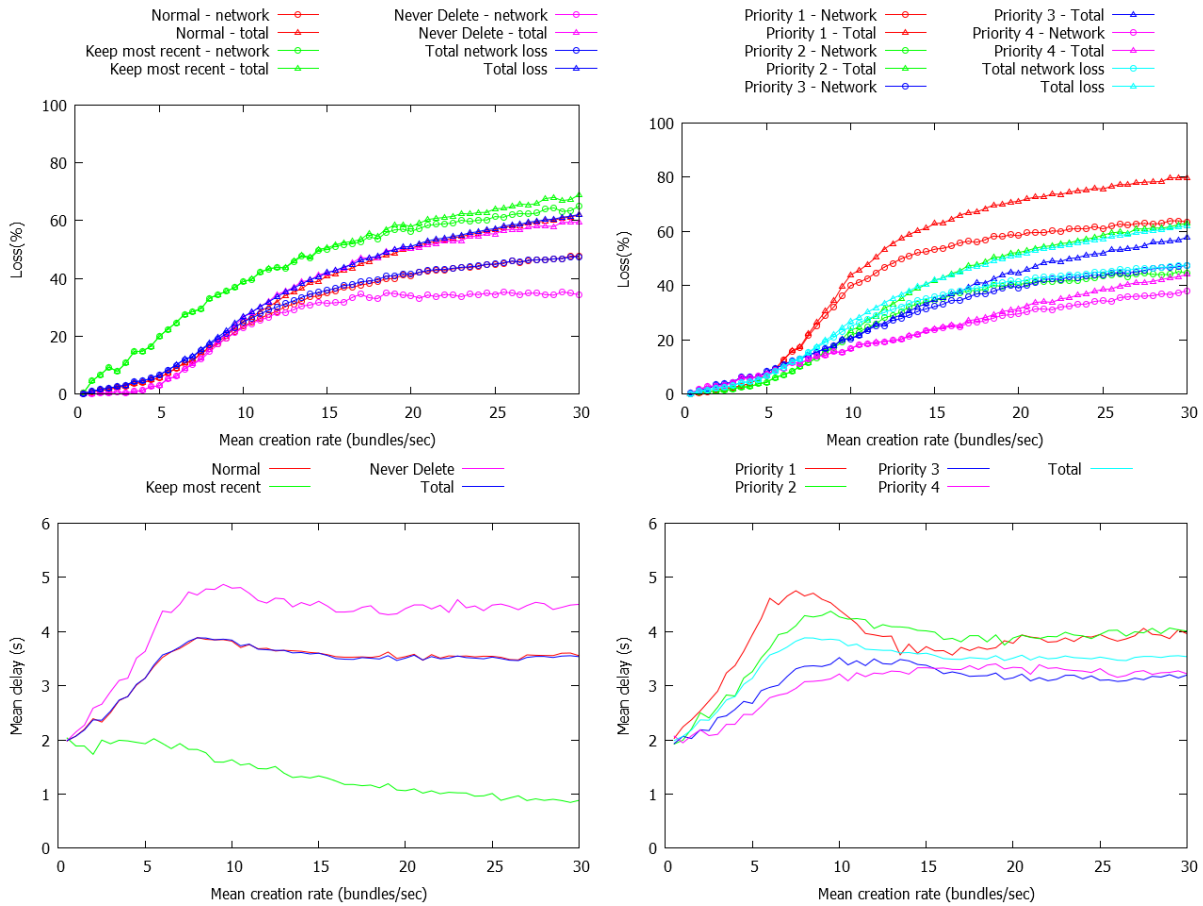


Figure 7.4: Loss and mean delay in ORWAR using the realistic mobility model
70% Normal, 10% Keep Most Recent, 20% Never Delete

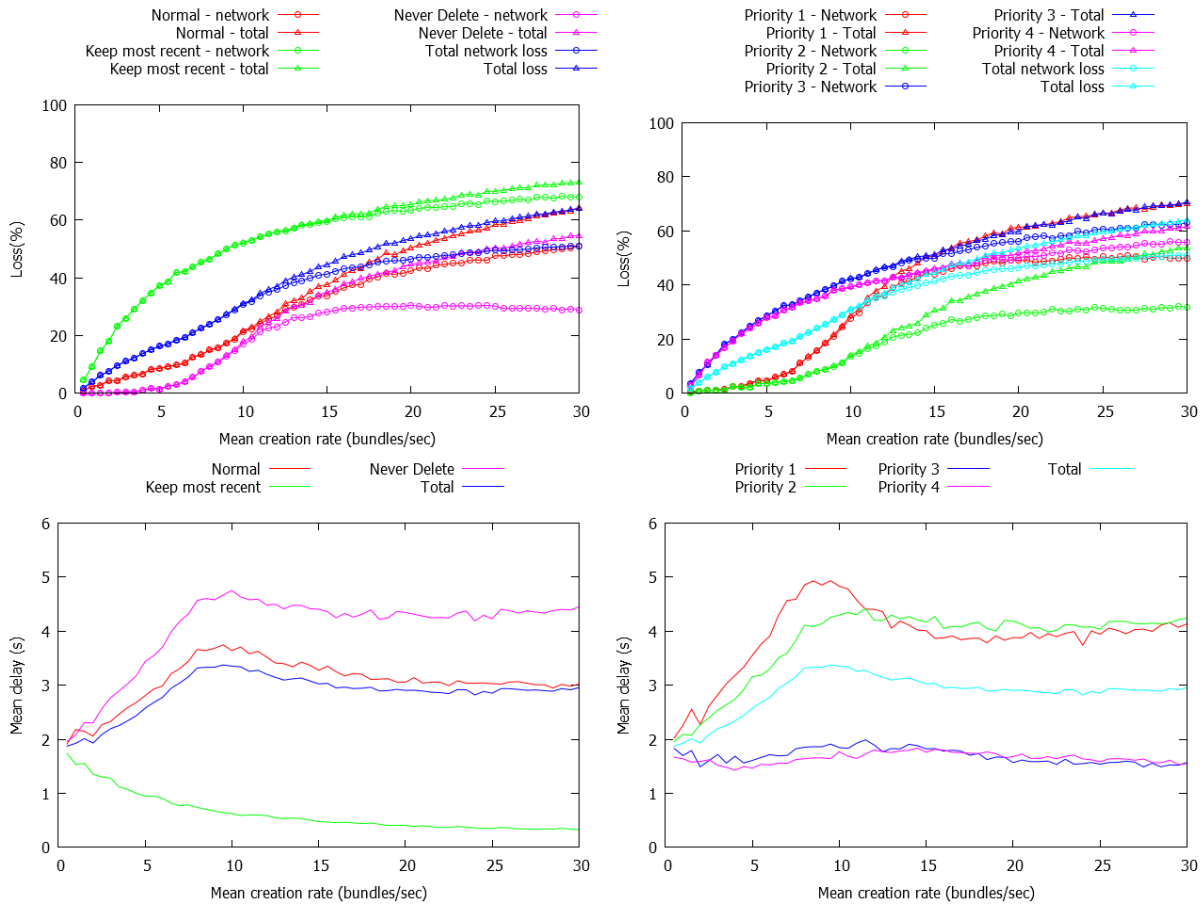


Figure 7.5: Loss and mean delay in ORWAR using the realistic mobility model
30% Normal, 35% Keep Most Recent, 35% Never Delete

transmitted with a low delay arrive at the destination.

Furthermore, comparing the delay per transmission priority to normal ORWAR, we also notice different behaviour. In normal ORWAR, as the creation rate increased, the delay per transmission priority converged to the mean delay measured over all traffic. When we add retention classes however, we notice a more distinct difference between high and low priority bundles. The highest priority bundles have a delay lower than the total traffic mean delay, while lower priorities have a higher value. Still, within the high- and low-priorities groups we see little difference in delay between the priorities, similar to when we compare all priorities in normal ORWAR.

What we therefore see is exactly what we required from the different retention classes. For low priority traffic, we see higher than average delay, though its loss is reduced. For high priority traffic we see exactly the opposite. Though we have some increased loss, delay for these classes is significantly reduced.

When we reduce the proportion of Normal bundles on the network, performance suffers. As shown in Figure 7.5, mean delay does not significantly improve if we add more Never Delete and Keep Most Recent bundles compared to Normal bundles. Loss however is drastically increased. There is almost no distinction between transmission priorities anymore in terms of loss. Especially at high creation rate values, Priority 1 bundles show almost identical total loss to Priority 3 bundles. However, the loss measured over all traffic has increased. This means that for no added benefit in terms of delay, more bundles are lost on the network. This supports our assumptions on which distributions of retention classes to use.

It should be noted however that these experiments are based on a small network with intermittent connectivity. As a network's size increases, or the number of applications/stream IDs changes, different parameters for Keep Most Recent and Never Delete traffic may be vital. In general, if the probability of replacing a bundle with a newer version using Keep Most Recent traffic is large, it may be better to reduce the number of bundles that can have this class. The values for Keep Most Recent we determined were based on 4 stream IDs which can contain Keep Most Recent bundles out of 16 total stream IDs. We have already shown that increasing Keep Most Recent traffic too much for these parameters will only increase loss and not reduce delay. At which proportion of Keep Most Recent traffic we see this tipping point is different for each network. We expect that as a network size increases, Keep Most Recent traffic becomes more effective to use. Bundles will generally have multiple paths available to them as the size increases. The probability of a bundle getting dropped completely from the network

is smaller as it is copied along multiple nodes. Keep Most Recent bundles can however aid in clearing unnecessary bufferspace of some nodes containing an older bundle. This older bundle may still be delivered through an alternate route however.

Similarly, the Never Delete proportion should be based on network connectivity, buffersize and bundlesize. Choosing a Never Delete proportion too high for these parameters will result in large loss as buffers will not be cleared fast enough. Also, as the network size increases, but moreover network density increases, the proportion of Never Delete bundles should be reduced. As there are more nodes to transmit traffic, the odds of a buffer filling up with bundles that cannot be cleared increase. Therefore, we assume that as a network's size and density increases, Keep Most Recent bundles should outnumber Never Delete bundles by a larger difference in proportions.

Of course, these results only describe part of the network's performance. We should also take throughput and/or arrival rate of bundles in consideration. Preferably, rather than measuring raw values we should relate this to different application streams or data types. This should give us a more accurate view whether the required performance is also met. For instance, even though we state that high priority traffic has reduced delay at the cost of increased loss, we do not know at what interval messages of a single stream arrive and if this is beneficial for the higher level application's performance. Fine-tuning of the proportion of Keep Most Recent and Never Delete traffic is therefore important for each individual scenario depending on the traffic and application requirements.

Chapter 8

Conclusions & Future work

8.1 Summary

In military networks, DTN can be used to significantly improve connectivity by dealing with some of the issues imposed on MANETs. To facilitate DTN functionality, special DTN replication based routing protocols have to be used along with the DTN bundle protocol. One such protocol is the ORWAR routing protocol. This protocol was chosen as a possible solution for routing within a military network as it attempts to optimize bandwidth use and limit power consumption.

Military networks contain several different data types, though all this traffic in general is transmitted to improve C2 in a military unit. Not all of this data can be transmitted using DTN, as some data types have too stringent delay requirements, such as voice communications

Based on these requirements, we define four transmission priorities which are used for bundle scheduling within ORWAR. Furthermore, we define three retention classes: Normal, Never Delete and Keep Most Recent. Normal bundles behave as if they have no special retention needs. These bundles will be removed if buffer storage should be freed to accommodate a higher priority bundle. Never Delete bundles do not affect other bundles already present upon insertion, but once inserted into a buffer, it may never be removed by any other incoming bundle. Keep Most Recent bundles are used to limit transmissions on the network of obsolete data, for instance obsolete position updates. These bundles will delete older related stored bundles from the buffer upon receiving.

To optimize bandwidth use, the ORWAR protocol calculates the contact window time between two nodes and schedules bundles accordingly as to minimize incomplete transmissions. This scheduling is based on the marginal utility of a bundle, defined as its utility per bit. Furthermore, it limits the number of copies of a bundle available on the network. To calculate the contact window time, nodes exchange geodata, acquired through for instance GPS signals.

To propagate bundle acknowledgements over the network, it stores bundle IDs in a KDM-vector. During handshaking, these vectors are exchanged between nodes and used to remove already arrived bundles from the buffer. The contents of these vectors will expire based on the TTL of the corresponding bundles. However, in addition to storing values in the KDM-vector for as long as the original bundle's TTL, a retention time τ is added. This value determines how long an acknowledgement should continue to exist in the KDM-vector when the original bundle would already have expired due to its TTL.

Our added retention classes have a significant impact on both the loss and delay of traffic on the network. Using Normal classed bundles, we would observe the different transmission priorities suffering more loss as the network load increases. This loss starts occurring at different loads for different priorities, with high priority messages able to suffer a far greater load before showing significant cost. In the meantime, the mean end-to-end delay of all traffic would stay roughly equal for increasing loads, presuming bundle sizes are variable.

Never Delete bundles, while their delivery accuracy is high, also increases the probability of later created/received bundles not being inserted in a node's buffer as these bundles will consume all available buffer storage and only free space once their TTL expires or an acknowledgement is received. In addition, all transmission priorities show roughly equal loss at every load. Even high priorities will suffer loss at lower load thresholds than when using only Normal classed bundles. Similar to Normal bundles however, the mean end-to-end delay would not vary much for increasing network load.

Keep Most Recent bundles show greatly increased loss as the network load increases. Every second a bundle stays in a buffer increases the probability of it being overwritten by a more recent bundle. Especially low transmission priorities will therefore suffer from increased loss. Only bundles that are transmitted fast from hop-to-hop, i.e. high transmission priority, do not suffer from the increased loss. As network load increases, the mean end-to-end delay is sharply reduced for Keep Most Recent bundles. At relatively high network loads, this behaviour results in only bundles with a high transmission priority which can be transmitted using an end-to-end connection being reliably transmitted. This goes against the principle of DTN where end-to-end connectivity should never be required.

The number of Keep Most Recent bundles which can effectively be used on a network seems to be be based on the network's size. As the network size increases, Keep Most Recent traffic becomes more and more effective to use. Using more Keep Most Recent bundles as the network size increases not only helps to improve the accuracy and delay of other bundles by removing obsolete buffer contents, but also does so at reduced risk of increased end-to-end loss.

One criticism of Keep Most Recent bundles however, is its tendency to let transmissions be for nought as the transmitted bundle is replaced by a newer version shortly after. This may be considered wasted bandwidth. However, this decreased performance only happens on a hop-by-hop basis. By replacing a bundle with a newer version, it saves potential resources down the line by not wasting it on transmissions of obsolete bundles.

Should this however become too much of a problem and end-to-end loss increases by too much as bundles will get replaced halfway during transmission, the application settings regarding the creation of these bundles should be adjusted. Keep Most Recent traffic in general is periodic information. By increasing the delay between two subsequent transmissions, end-to-end accuracy will be improved. For instance, if position updates are injected every second, but only arrive every five seconds due to obsolete updates being replaced halfway, these updates can also be transmitted once every five seconds. Fine tuning all settings for each individual network is therefore very important.

All in all, when comparing the normal ORWAR protocol to our QoS-enabled ORWAR protocol using appropriate traffic distributions, we notice that the loss of all measured traffic is roughly equal, though mean delay is significantly reduced. This conclusion however does not provide us with the total picture as for a true performance evaluation we need to also take into account the actual bandwidth use and application-specific requirements to decide whether some of the proposed trade-offs are worth it. However, it does show us that the current limited QoS provisioning in ORWAR using marginal utilities can be improved.

8.2 Conclusions

During this research we were able to make some discoveries regarding the ORWAR protocol and its performance on the network. Similarly, our designed retention classes have significant impact on the network performance compared to ORWAR without support for these classes. We can therefore split our conclusions into different affected areas.

8.2.1 Traffic class definitions and distributions

- Traffic on military networks that can be transmitted using DTN can be split in Command data, Situational Awareness data and Reports. Each traffic type however, poses different requirements on the network in regards to transmission and retention.
- To support traffic classes in network, ORWAR's inherent utility values are replaced by transmission priorities. Furthermore, its receive algorithm is edited to provide functionality for the different retention classes.
- For this purpose, we also define a new bundle block which stores a bundle's stream ID to ensure only related bundles are removed by Keep Most Recent bundles. In addition, this bundle block also stores the number of copies remaining of a bundle.
- We recommend using the Never Delete retention class only for low-priority messages and the Keep Most Recent class for high-priority messages.
- The proportion of all traffic which can have a specific retention class should be fine tuned for every specific network. Using the wrong proportions will have a detrimental effect on performance.
- In our simulations, we found that Normal-classed bundles should still outnumber the combination of Never Delete and Keep Most Recent bundles.
- Keep Most Recent bundles should in turn outnumber Never Delete bundles. The number of Keep Most Recent bundles which can effectively be used on a network seems to be be based on the network's size. As the network size increases, Keep Most Recent traffic becomes more and more effective to use.
- The number of Never Delete bundles on a network should be kept relatively low depending on a buffer's storage capacity. These bundles will possible occupy a node's buffer for a long period of time, wasting storage resources.

8.2.2 General ORWAR performance

- ORWAR is a good routing protocol choice for a network that is very sparsely connected with frequent topology changes, provided some elements are improved.
- The retention of acknowledgements allows the acknowledgement propagation to also be delay-tolerant, providing the source node with a means to receive an acknowledgement, even though the bundle itself may already have expired.
- The protocol as it is currently researched has its share of limitations:
  - Neighbours' buffer contents unknown.
  - Contact window use limitations
  - Limited buffer maintenance
  - Limited acknowledgement support
- In addition, ORWAR as it currently stands has no support for multiple nodes sharing a medium at the same time, its associated issues are:
  - Different contact windows for different neighbours
  - Different buffer contents of different neighbours
  - Insufficient acknowledgement propagation
  - Unable to determine next-hop node
- To overcome these issues, the following improvements have been made to the ORWAR protocol:
  - Exchange of buffer contents during handshaking, in case a node has multiple neighbours, a threshold is used to determine which bundles should be transmitted.
  - Periodic recalculation of contact windows with current neighbours. When a node has multiple neighbours, it uses the largest available contact window to determine its transmission buffer. A new contact window will only be used if it differs from the old window by a certain threshold. Lastly, an *effective* contact window is introduced to deal with multiple nodes sharing one medium.
  - Before transmissions, a bundle's TTL will be checked, not just during handshaking.
  - We propose the use of negative acknowledgements to prevent retransmissions of bundles rejected by the receiver. Furthermore, KDM-vectors are both periodically exchanged and requested again by a node from its neighbours when it reorders its transmission buffer. An acknowledgement will also be returned by an intermittent node if it receives a bundle it already has stored in its KDM. Lastly, acknowledgements are transmitted using single-hop broadcasts to inform multiple nodes about a bundle's arrival.
  - All transmissions to intermittent nodes will be done using single-hop broadcasts to promote propagation and remove the problem of determining the next-hop node of a bundle.
- When observing the behaviour of ORWAR in different networks, we found node mobility, network transmission rate, node buffersize and bundle size, along with a bundle's TTL to influence the performance of different transmission and retention classes.
- It is evident that ORWAR is designed for networks with nodes only having small contact windows. When the contact window time increases, or when nodes are stationary, we notice that the performance of the originally researched ORWAR suffers.
- Contact window time estimation needs not to be very accurate when the average bundlesize is significantly smaller than the maximum amount of data that can be transmitted during one contact window. However using bundles of such small size can result in unnecessary overhead. This will waste bandwidth and therefore a bundlesize should be fine tuned to the networking needs to minimize incomplete transmissions without adding too much overhead. When optimizing bundle size however, aside from the influence of node mobility and transmission rate, one also would need to factor in whether application messages fragmented into many bundles can eventually be reconstructed successfully at the destination. It is not feasible to fragment messages to reduce the probability of incomplete transmissions if not enough fragments will eventually reach the destination.

8.2.3 Performance of ORWAR with traffic classes

- Deleting obsolete bundles significantly improves the performance of bundles unaffected by its mechanisms, due to being classed Never Delete or not having the same stream ID. As bundles are constantly deleted from the buffer, other bundles may be transmitted at a sooner opportunity than before, improving both their delivery accuracy and delay.
- When using the suggested traffic proportions in a realistic military network running the QoS-enabled ORWAR protocol, we notice some differences compared to the original ORWAR protocol.
  - As load increases, the measured loss of all transmission priorities deviate less from the mean accuracy than when using normal ORWAR.
  - Delay however is increased for lower-priority messages and decreased for higher-priority messages when compared to the mean delay.
  - In addition, the mean delay for all bundles is significantly lower than in the original ORWAR as buffer contents are reduced through the use of Keep Most Recent traffic, reducing waiting time in a buffer.
- Frequency of inserted Keep Most Recent traffic should be adjusted if too many bundles get removed during transit, reducing the delivery frequency of this traffic.
- Loss of all measured traffic is roughly equal, though mean delay is significantly reduced.

8.3 Research Questions

At the start of this thesis, we have set up some research questions to answer. The main question asked was

> *Can we improve ORWAR to better support QoS provisioning for use in military MANETs?*

Where we defined the following sub-questions:

1. What traffic classes can be distinguished in military MANETs and what requirements do they impose?
2. Does the current design of ORWAR support a separation in transmission priority QoS and buffer management QoS?
3. How can the required QoS be implemented in ORWAR?
4. How does the implemented QoS perform in comparison to non-QoS ORWAR?

These questions will be answered below.

> *What traffic classes can be distinguished in military MANETs and what requirements do they impose?*

In military MANETs we can distinguish several kinds of data to improve C2 in a military unit. This data can be divided in three main groups:

- Command Data
- Situational Awareness Data
- Reports

The data in these groups can take many forms, from database updates to text and imagery. Different traffic types, even within one of the above groups, can impose different requirements. These requirements can be divided into some traditional metrics as priority, delay-tolerance and required delivery accuracy. In addition to these traditional metrics however, we can also add another stating whether the receiver is only interested in the most recent transmission of a source. This is the case for periodic updates and location information such as BFT where the receiver is only interested in the most recent position update or database synchronization. This extra metric is however only relevant to DTN as transmissions will not always occur end-to-end. When transmissions occur end-to-end, the source may choose to stop transmission of obsolete data as the application generates newer data. When they do not occur end-to-end however, intermediate nodes also need to know if obsolete data should be further transmitted once a path to the destination becomes available again.

To accommodate all the above requirements, we have split the traffic class in a transmission priority and retention class. The transmission priority determines which bundle to transmit first and can play

a factor in deciding which bundles to delete should a storage buffer reach its maximum capacity. The retention class describes the behaviour within a node's storage buffer. We have defined three retention classes: Normal, Never Delete and Keep Most Recent. Normal bundles can always be deleted by bundles with a higher priority-per-bit or a more recent Keep Most Recent bundle. Never Delete bundles can only be removed from a buffer by their TTL expiring or upon receiving an acknowledgement. This class is useful for bundles with high delay-tolerance but also a high required delivery accuracy. Keep Most Recent bundles serve the purpose of deleting obsolete data. As soon as a Keep Most Recent bundle is received, obsolete bundles belonging to the same traffic flow are removed from storage. This class will generally be used for periodic updates and synchronizations.

Does the current design of ORWAR support a separation in transmission priority QoS and buffer management QoS?

The current design of ORWAR has some limitations which need to be addressed before it can be applied to a practical scenario such as military MANETs before addressing QoS support, mentioned in Section 8.2.2. Once these issues are addressed however, ORWAR provides a decent framework to add our intended separation of transmission priority and buffer management. The implementation of transmission priorities is easy as ORWAR already assigns a utility value to bundles which is used for priority scheduling. By using this utility value to define our transmission priorities, we can order bundles within the buffer accordingly. ORWAR offers no inherent support for the retention classes, however by making some modifications to the receive algorithm we can add this support, mentioned below.

How can the required QoS be implemented in ORWAR?

As we stated above, transmission priorities can be implemented through the use of ORWAR's existing utility values. To add support for retention classes, the receive algorithm needs to be modified. First of all, ORWAR needs to detect the retention class of a bundle, which will be stored in a bundle's header information. When ORWAR is looking for bundles to delete as the storage buffer reaches maximum capacity, it should ignore bundles marked as Never Delete. Furthermore, when it receives a Keep Most Recent bundle, it should first check if enough space can be freed by deleting bundles which would be obsoleted by the incoming bundle. These obsoleted bundles need to be deleted upon accepting a Keep Most Recent bundle, regardless of whether the bundle had reached maximum capacity before accepting the bundle.

To support the deleting of obsolete bundles, we added a stream ID to bundles which identifies the data flow they belong to, to make sure Keep Most Recent bundles can only delete relevant obsolete bundles. This stream ID is also needed to prevent a node from accepting obsolete bundles. As nodes forward copies of bundles rather than the bundle itself, it is possible for a bundle to receive a bundle it has already deleted as an obsolete bundle from another intermittent node. By storing the creation timestamp of the most recent received Keep Most Recent bundle for a given stream ID, a node can reject obsolete incoming bundles. It should however be noted that even though a Keep Most Recent bundle will remove obsolete bundles and cause the node to reject future incoming obsolete bundles, it cannot remove or cause the rejection of Never Delete bundles.

How does the implemented QoS perform in comparison to non-QoS ORWAR?

We see some significant differences in behaviour between QoS-enabled and non-QoS ORWAR. How the QoS-enabled ORWAR performs on a network is however very dependent on the distribution of retention classes on the network. This distribution should be fine-tuned to the requirements of a network or performance will suffer. The advantage of Never Delete bundles is their very high delivery accuracy which is especially advantageous to low-priority bundles. However this comes at the cost of reduced storage efficiency. Should the number of Never Delete bundles be too high, loss on the network will increase as nodes will be forced to reject incoming or newly created bundles due to insufficient storage available.

Keep Most Recent bundles on the other hand are very effective for storage and ensures that those bundles that do get delivered, get delivered with little delay. This however comes at the cost of increased loss and a bad configuration of number of Keep Most Recent bundles on the network will be detrimental to performance as less bundles will reach their final destination as they will get deleted somewhere during transmission. The advantage of deleting obsolete bundles before reaching the final destination however is that it frees up node resources for other bundles unaffected by the incoming Keep Most Recent bundle, improving both their delivery accuracy and delay.

When using a distribution of retention classes deemed advantageous to our specific network, we see a change in both loss and delay compared to non-QoS ORWAR. In non-QoS ORWAR, different transmission priorities would arrive with little variation in delay, but a larger variation in loss, with low priority messages suffering greater losses as network load increases. When adding our QoS, we see the opposite happen. While low priority messages still suffer a greater loss than high priority messages, all priorities are closer to the mean loss of all priorities combined. This means that high priority messages

suffer a greater loss in general than non-QoS normal. This extra loss however is caused by bundles being removed somewhere during transmission through Keep Most Recent bundles.

The mean delay of all transmission priorities however is reduced when using QoS-enabled ORWAR. As Keep Most Recent bundles delete obsolete bundles from intermittent nodes, resources are freed for other bundles. Furthermore, while in non-QoS ORWAR the delay was roughly equal for all transmission priorities, we see a clear distinction between high and low priority bundles in terms of delay in the QoS-enabled version. Here low-priority bundles will have a significantly longer delay than high-priority bundles. This is caused by the Never Delete bundles, which improves a low priority's accuracy, i.e. reducing loss, however it does not improve its priority. It therefore may have to wait longer in a buffer before it is transmitted, but has a larger probability of actually being transmitted compared to non-QoS ORWAR.

This leads us back to the main question:

Can we improve ORWAR to better support QoS provisioning for use in military MANETs?

It is possible to adapt ORWAR to support QoS provisioning, however it is unclear whether or not it is an improvement in relation to the military MANET's requirements. While in terms of accuracy and delay it is an improvement, other factors need to be taken into consideration as well. For instance, we have not determined whether an individual application's performance has improved through this QoS provisioning. In addition, other metrics need to be taken into account as well, such as network throughput and goodput.

However, based on the results we have generated in this research, we can state that in terms of accuracy and delay we have improved upon ORWAR to better suit the requirements of military MANETs. In addition, other than applying our traffic classes we have shown how ORWAR can be significantly improved to function in a practical scenario.

8.4 Future Work

While we have attempted to improve upon ORWAR by adding QoS using transmission priorities and retention classes, there is still a lot left to do. We have defined some parameters and protocol behaviour, though these choices may not always have been optimal. The three main issues that should be validated are the decision in which contact window to use, the contact window threshold and the neighbour's buffer contents threshold. We have chosen to use the largest available contact window and the thresholds based on reasoning and estimating fair values, however further experiments should be performed to determine their influence on performance and obtain optimal values.

We have used our neighbour's buffer contents threshold to determine whether or not a bundle should be transmitted. This threshold will become more lenient if a node cannot fill its transmission buffer to make use of its full contact window. Being too lenient with the required threshold to be added to the transmission buffer can be suboptimal. When increasing the leniency of the threshold, a node may be using the medium for a bundle already well-propagated over the network while one of its neighbours may have a bundle waiting which has not propagated at all. It might therefore be better not to use the medium for the well-propagated bundle but allow the new bundle to be transmitted instead.

Not adjusting the threshold and being too strict however, may result in the medium not being used at all and/or bundles not being propagated enough. If nodes may not have enough bundles to add to its transmission buffer to fill its contact window due to the threshold, the medium will stay idle. Increasing the leniency of the threshold will however result in the medium being used. Furthermore, if a node will communicate very often with a subset of neighbouring nodes, it may result in bundles not being propagated enough as they will not meet the threshold when these neighbours are in range and already have this bundle. An optimal algorithm should be developed to determine the threshold in different situations.

In ORWAR, the number of copies of a message is limited and when transmitting a bundle, a sender sends half the copies to a neighbouring node. When broadcasting bundles rather than unicasting, if a node broadcasts half the copies it has, after the transmission there may be even more copies on the network than before transmitting. This may reduce performance as the network increases in size and also reduce storage efficiency. Reducing the number of copies transmitted based on the number of neighbours, for instance broadcasting $^1/_3$ of the copies while keeping $^1/_3$ when a node has 2 neighbours might result in inefficient propagation and reduced delivery accuracy. Further research should determine a proper distribution of copies on the network.

In addition, the relation between the number of copies and the bundle's utility was researched for three different utility values. When using four priorities, a better optimum should be determined. In our simulations this has not been an issue as the simulated network was too small to limit the number of copies on the network.

We have also made some assumptions regarding the distribution of retention classes on the network based on preliminary research. These distributions seemed to increase performance on our specific

network model. When adding more nodes, different traffic flows and different mobility models, the distribution should change. Further research may determine an algorithm to optimize these distributions per network.

Lastly, we have stated that this research does not provide the complete picture in terms of how much an improvement it offers on non-QoS ORWAR. Aside from our used metrics, different metrics should also be checked and compared to the requirements. Some examples include the throughput/goodput on the network and application performance. DTN tends to add more aspects and accompanying requirements to normal MANET networking, e.g. the storage efficiency of a node. Only by analysing the various aspects of network performance in DTN can an overall conclusion whether the performance is improved be reached.

Bibliography

[1] United States Department of Defense, “Joint doctrine for information operations,” October 1998.

[2] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, “Delay-tolerant networking architecture,” RFC4838, April 2007.

[3] K. Fall, “A delay-tolerant network architecture for challenged internets,” in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM ’03. New York, NY, USA: ACM, 2003, pp. 27–34.

[4] “The InterPlaNetary (IPN) Internet Project,” as described on the Internet Society’s InterPlanetary Networking Special Interest Group (IPNSIG) official site. [Online]. Available: http://www.ipnsig.org

[5] K. Scott and S. Burleigh, “Bundle protocol specification,” RFC5050, November 2007.

[6] T. Clausen and P. Jacquet, “Optimized link state routing protocol (OLSR),” RFC3626, October 2003.

[7] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc on-demand distance vector (AODV) routing,” RFC3561, July 2003.

[8] A. Vahdat and D. Becker, “Epidemic routing for partially connected ad hoc networks,” Department of Computer Science, Duke University, Tech. Rep. CS-2000-06, April 2000.

[9] A. Lindgren, A. Doria, and O. Schelén, “Probabilistic routing in intermittently connected networks,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 7, no. 3, pp. 19–20, Jul. 2003.

[10] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, “Maxprop: Routing for vehicle-based disruption-tolerant networks,” in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, 2006, pp. 1–11.

[11] A. Balasubramanian, B. Levine, and A. Venkataramani, “DTN routing as a resource allocation problem,” in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM ’07. New York, NY, USA: ACM, 2007, pp. 373–384.

[12] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, “Spray and wait: an efficient routing scheme for intermittently connected mobile networks,” in *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, ser. WDTN ’05. New York, NY, USA: ACM, 2005, pp. 252–259.

[13] G. Sandulescu and S. Nadjm-Tehrani, “Opportunistic DTN routing with window-aware adaptive replication,” in *Proceedings of the 4th Asian Conference on Internet Engineering*, ser. AINTEC ’08. New York, NY, USA: ACM, 2008, pp. 103–112.

[14] G. Kamphuis, “Disruption tolerant networking in a heterogeneous military MANET environment,” University of Twente, October 2013.

[15] A. Lindgren, A. Doria, and O. Schelén, “Probabilistic routing in intermittently connected networks,” in *Service Assurance with Partial and Intermittent Resources*, ser. Lecture Notes in Computer Science, 2004, vol. 3126, pp. 239–254.

[16] D. S. Alberts, *Information age transformation: Getting to a 21st century military*. Department of Defense Command and Control Research Program, June 2002.

[17] D. Anzaldi, “ORWAR: a delay-tolerant protocol implemented on the android platform,” Master’s thesis, Linköpings universitet, December 2010.

[18] D. Ellard, R. Altmann, A. Gladd, and D. Brown, “DTN IP neighbor discovery (IPND),” draft-irtf-dtnrg-ipnd-02 - work in progress, November 2012.

[19] G. Sandulescu and S. Nadjm-Tehrani, “Adding redundancy to replication in window-aware delay-tolerant routing.” *Journal of Communications*, vol. 5, no. 2, 2010.

[20] A. Roodselaar, “Disruption tolerant networking in tactical communication networks,” University of Amsterdam, June 2011.

[21] X. Lu, Y.-C. Chen, I. Leung, Z. Xiong, and P. Liò, “A novel mobility model from a heterogeneous military manet trace,” in *Proceedings of the 7th international conference on Ad-hoc, Mobile and Wireless Networks*, ser. ADHOC-NOW ’08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 463–474.

[22] H. Seo, S. Kim, and J. Ma, “A novel mobility model for the military operations with real traces,” in *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*, vol. 1, 2010, pp. 129–133.